```
*****************************************************************************
*Calendar Program / Script
*
*****************************************************************************
*Main
*-----
* See what was passed as arguements and then do it
*       Options
*   -------
*   day print - then exit
*   input calendar invite then exit
*   engage graphical user interface
* exit
*****************************************************************************
*Day Print
*----------
* Input Calendar(s)
* Determine day schedule and which calendars to print
* Print calendar
* Exit
*****************************************************************************
* Calendar invite input
*---------------------
* Input invite
* Input calendars
* Add invite to appropriate calendar
* ouptput updated calendar
* exit
*****************************************************************************
=============================================================================================…
/*************************************************************************/
/* Etyme.h                                                               */
/* --------------------------------------------------------------------- */
/* Version 0.0  00/2000 eab                                              */
/* --------------------------------------------------------------------- */
/* Purpose                                                               */
/* --------------------------------------------------------------------- */
/* Platform: all                                                         */
/* --------------------------------------------------------------------- */
/* History:                                                              */
/* --------------------------------------------------------------------- */
/*************************************************************************/

/*************************************************************************/
/********************              #includes              ********************/
/*************************************************************************/


@interface

/*************************************************************************/
/********************     member declarations              ********************/
/*************************************************************************/
{
struct edaet {        /* Internal Earth Date and Time GMT*/
    float   seconds;  /* seconds from midnight  0->86399 */
    short   days;     /* days in year 1->365 or 366      */
    int     year;     /* year  (Gregorian based)         */
} edaet;

struct etyme {        /* standard 24 hour Earth Clock    */
        float    seconds;    /* 0-59 seconds in current minute  */
        short    minutes;    /* 0-59 minutes in current hour    */
        short    hour;    /* 0-23 hour in current day        */
```

```
} etyme;

struct atyme {        /* angular Earth Clock    */
        float    arc_seconds;    /* 0-59 seconds in current minute  */
        short    arc_minutes;    /* 0-59 minutes in current hour    */
        short    degrees; /* 0-359 degree in current day     */
} atyme;

struct gdates {                  /* Gregorian Date - std 24hr clock */
    struct  etyme     tyme;      /* time of day - 24 hour clock */
    short   day;                 /* day in month     1-31           */
    int     month;               /* month in year    1-12           */
    int     year;                /* year             +-nnnnn  CE    */
    float   zone;                /* local time zone  -12 -> + 12    */
    int     diwk;                /* day in week      1-7            */
} gdates;

struct gdatea {                  /* Gregorian Date - angular 24hr clock */
    struct  atyme     tyme;      /* time of day - angular clock */
    short   day;                 /* day in month     1-31               */
    int     month;               /* month in year    1-12               */
    int     year;                /* year            +-nnnnn CE          */
    int     diwk;                /* day in week     1-7                 */
} gdatea;

struct hdates {                  /* Hobbit Date - std 24hr clock */
    struct  etyme     tyme;      /* time of day - 24 hour clock */
    short   day;                 /* day in month     1-30           */
    int     month;               /* month in year    0-12           */
    int     year;                /* year             +-nnnnn  CE    */
    float   zone;                /* local time zone  -12 -> + 12    */
    int     diwk;                /* day in week      0-7            */
} hdates;

struct hdatea {                  /* Hobbit Date - angular 24hr clock    */
    struct  atyme     tyme;      /* time of day - angular clock */
    short   day;                 /* day in month     1-30               */
    int     month;               /* month in year    0-12               */
    int     year;                /* year            +-nnnnn CE          */
    int     diwk;                /* day in week      0-7                */
} hdatea;

}
/*****************************************************************************/
/*******************     method declarations              *******************/
/*****************************************************************************/


@end

/*****************************************************************************/
/*******************     method   descriptions            *******************/
/*****************************************************************************/

/*****************************************************************************/
/* method                                                                    */
/* ------------------------------------------------------------------------- */
/* Purpose:                                                                  */
/* Version:                                                                  */
/* Author:                                                                   */
/* Date:                                                                     */
/* ------------------------------------------------------------------------- */
/* Input:                                                                    */
/* Returns:                                                                  */
/* Other:                                                                    */
/                                                                            */
```

```
/* Functions Used (with module/library):
/*       Function                - module or library                        */
/* ---------------------------------------------------------------------- */
/* Operation:
/* ---------------------------------------------------------------------- */
/* Version history:
/* Version - Date    - Author        - Notes                               */
/*************************************************************************/

/*************************************************************************/
/********************                end                   ********************/
/*************************************************************************/
```

```
/*************************************************************************/
/* Calendar.h
/*-----------------------------------------------------------------------*/
/* current v1.00 July 2004
/*=======================================================================*/
/*   Purpose
/*       This program is
/*
/*
/* ---------------------------------------------------------------------- */
/*   Platform:  all
/*-----------------------------------------------------------------------*/
/*   History
/*   -------
                                                                          */

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

/*************************************************************************/
/************************     #includes     ********************************/
/*************************************************************************/
#include ".h"

/*************************************************************************/
/************************   typedef, define enum   ***************************/
/*************************************************************************/

#define BackSize         50         /* size of backtrack buffer originally 20 */

/*************************************************************************/
/************************   function prototypes    ***************************/
/*************************************************************************/

/* Public - used by other modules */
/*-------------------------------*/

/* Private - for use by this module only */
/*--------------------------------------*/

/*************************************************************************/
/************************   function descriptions    *************************/
/*************************************************************************/

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

```
/*************************     Public functions       *************************/
/*****************************************************************************/

/*****************************************************************************/
/* module                                                                    */
/* ------------------------------------------------------------------------- */
/* Purpose:                                                                  */
/* Version:                                                                  */
/* Author:                                                                   */
/* Date:                                                                     */
/* ------------------------------------------------------------------------- */
/* Input:                                                                    */
/* Returns:                                                                  */
/* Other:                                                                    */
/* Functions Used (with module/library):                                     */
/*       Function                - module or library                         */
/* ------------------------------------------------------------------------- */
/* Operation:                                                                */
/* ------------------------------------------------------------------------- */
/* Version history:                                                          */
/* Version - Date    - Author        - Notes                                 */
/*****************************************************************************/

/*****************************************************************************/
/*************************    Private functions       *************************/
/*****************************************************************************/



/*****************************************************************************/
/* module                                                                    */
/* ------------------------------------------------------------------------- */
/* Purpose:                                                                  */
/* Version:                                                                  */
/* Author:                                                                   */
/* Date:                                                                     */
/* ------------------------------------------------------------------------- */
/* Input:                                                                    */
/* Returns:                                                                  */
/* Other:                                                                    */
/* Functions Used (with module/library):                                     */
/*       Function                - module or library                         */
/* ------------------------------------------------------------------------- */
/* Operation:                                                                */
/* ------------------------------------------------------------------------- */
/* Version history:                                                          */
/* Version - Date    - Author        - Notes                                 */
/*****************************************************************************/
```

Calendar Program

This program has a number of different functions

1) Convert between any two calendar systems that I know
2) Display a calendar period (month, week, day) of a desired system
3) Appointments
4) Compute delta date/time

Calendar systems

- Julian
- Gregorian
- Shire

```
    'Scientific'-a
    Mars
    Chinese-learn and code

Structures
    jdate - secs from midnight
            day from year start
            year
            time zone delta hour from Zulu



/* ===== * ===== * ===== * ===== ****** ===== * ===== * ===== * ===== * ===== */
/* Calendar systems                                                          */
/* ----------------                                                          */
/* Internal                                                                  */
/*       Year - Day - Time                                                   */
/*                                                                           */
/*     Year - could be Christian or some other - pick is Christian           */
/*     Day - First day of the year is 1 and then sequentially numbered till  */
/*           end of year                                                     */
/*     Time - Earth 24 hour day at Zulu (Greenwich Mean Time)                */
/*            24 hours/day, 60 minutes/hour, 60 seconds/minute               */
/* ----- -----                                                               */
/* Gregorian                                                                 */
/*       Year - Month - Day of Month - Time - time zone - Day of Week        */
/*              This calendar was started in year 1582 A.D. with a date shift */
/*                   from the existing calendar so need to put that in the   */
/*                   algorithm.  10/5/1582==10/15/1582                       */
/*                                                                           */
/*       Year - Christian calendar wih A.D. and B.C.                         */
/*       Month - 12 months of varying days, second month changes number of   */
/*               days if leap year                                           */
/*            January-31, February-28?29, March-31, April-30, May-31, June-30 */
/*            July-31, August-31, September-30, October-31, November-30,     */
/*            December-31                                                    */
/*       Day of Month - see above for days in each month                     */
/*       Time - Earth 24 hour time                                           */
/*       Time Zone - ranges from -12 to +12 with 0 being Zulu                */
/*       Day of Week - 7 days per week,  each year starts on a different day */
/*            Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday */
/* ----- -----                                                               */
/* Shire  Hobbit                                                             */
/*       Year - Month - Day of Month - Time - time zone - Day of Week        */
/*                                                                           */
/*       Year - ?                                                            */
/*       Month - 12 months of 30 days - 5 days not part of any month or week */
/*            Afteryule, Solmath, Rethe, Astron, Thrimidge, Forelithe,       */
/*            Afterlithe, Wedmath, Halimath, Winterfilth, Blotmath, Foreyule */
/*       Day of month - 1 to 30 with a few days not part of any month        */
/*       Time - Earth 24 hour time                                           */
/*       Time Zone - ranges from -12 to +12 with 0 being Zulu                */
/*       Day of Week - 7 days per week, each year starts on Sterrendei       */
/*            Sterrendei, Sunnendei, Momendei, Trewesdei, Hevenesdei,        */
/*            Meresdei, Highdei                                              */
/*       Year's progression                                                  */
/*            2Yule, Afteryule, Solmath, Rethe, Astron, Thrimidge, Forelithe, */
/*            1Lithe,                                                        */
/*            Midyear's Day, (OverLithe), - these two are not part of the week*/
/*            2Lithe, Afterlithe, Wedmath, Halimath, Winterfilth, Blotmath,  */
/*            Foreyule, 1Yule                                                */
/* ----- -----                                                               */
/* International Fixed                                                       */
```

```
/*      Year - Month - Day of Month - Time - time zone - Day of Week        */
/*                                                                          */
/*      Year - ?                                                            */
/*      Month - 13 months
/*      Day of month - 1 through 28 with MidDay and LeapDay not part of month*/
/*      Time - Earth 24 hour time                                           */
/*      Time Zone - ranges from -12 to +12 with 0 being Zulu                */
/*      Day of Week - 7 days per week,  each year starts on Sunday          */
/*           Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday  */
/*           MidDay and LeapDay not part of any week                        */
/*           MidDay is in the 7th month between the 14th and 15th           */
/*           LeapDay is with MidDay on leap years                           */
/* ----- -----                                                              */
/* perpetual
/*      Year - Month - Day of Month - Time - time zone - Day of Week        */
/*                                                                          */
/*      Year - ?                                                            */
/*      Month - 12 months 8-30 4-31
/*      Day of month - 1 through 28 with MidDay and LeapDay not part of month*/
/*      Time - Earth 24 hour time                                           */
/*      Time Zone - ranges from -12 to +12 with 0 being Zulu                */
/*      Day of Week - 7 days per week,  each year starts on Sunday          */
/*           Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday  */
/*           MidDay and LeapDay not part of any week                        */
/*           WorldDay is last day of year except every 4th year             */
/*           LeapDay is with MidDay on leap years                           */
/* ----- -----                                                              */
/* Mars1
/*  MDay = 24 hours, 39 minutes, 35 seconds of ETime
/*  MYear =  668.62... MDays    = 687 EDays

Structures
    jdate - secs from midnight
            day from year start
            year
            time zone delta hour from Zulu



Years divisible by 4 a leap year except divisible by 100 unless divisible by 400

*****************************************************************************
Internal date:time format is Julian GMT -  year:day:sssss.sss
time zones internal sssss
/* */
sub time2zulu { }
/* This subroutine converts a zone julian date:time to zulu given original date:time and time zo…
Algorithm:
        add time to original
        if time > 86400 then subtract 86400 from time and increment date
        if time < 0000 then add 2400 to time and decrement date
        if day > year's end then date=1 and increment year
        if day < 1 then day=1 and decrement year
/* */
sub zulu2time { }
This subroutine converts a zulu date:time to zone given zulu date:time and time zone
Algorithm:
        subtract time from zulu
        if time > 2400 then subtract 2400 from time and increment date
        if time < 0000 then add 2400 to time and decrement date
        if day > year's end then date=1 and increment year
        if day < 1 then day=1 and decrement year
/* */
int Gregorian2Julian();
```

```
/* This subroutine converts a Gregorian date to a Julian date                    */
Algorithm:
        Determine if leap year and use appropriate month set
        determine integer month
        return day = day of month plus sum of previous months

sub Julian2Gregorian { }
This subroutine converts a Julian date to a Gregorian date
Algorithm:
        Determine if leap year and use appropriate month set
        While day !inMonth increment month
        Return Gregorian Date
/* */
sub Shire2Julian { }
This subroutine converts a Shire date to a Julian date
Algorithm:
        If special day then set value and return day
        Determine if leap year and use appropriate year set
        determine integer month
        return day = day of month plus sum of previous months
/* */
sub Julian2Shire { }
This subroutine converts a Julian date to a Shire date
Algorithm:
        Determine if leap year and use appropriate year set
        If special day then return Shire date by table
        While day !inMonth increment Month
        return Shire date
/* */
sub ShireDayofWeek { }
This subroutine returns the number of the day of week 0:7 given a Shire date
Algorithm:
/* */
sub GregorianDayofWeek { }
This subroutine returns the number of the day of week 1:7 given a Gregorian Date
Algorithm:
/* */
sub isLeap { }
This returns True if LeapYear otherwise False
Algorithm:
        if year mod 4  0 then return False
        if year mod 4 = 0 and year mod 400 = 0 return True
        if year mod 4 = 0 and year mod 100  0 return True else False
/* */
sub time2secs { }
This subroutine converts 24 hour time to seconds from midnight
Algorithm:
/* */
sub secs2time ( )
This subroutine converts seconds from midnight to 24 hour time
Algorithm:
        ss.sss = secs mod 60
        hour = trunc(secs - ss.sss)/3600
        minutes = ((secs - ss.sss) - (hour*3600))/60
/* */
sub 12convert24
This sub converts a 24 hour time to 12 hour time
Algorithm:
        if hh < 13 return time + 'a.m.'
                else hh -= 12 and return time + 'p.m.'
/* */
sub 24convert12
This sub converts a 12 hour time to 24 hour time
Algorithm:
        if 'a.m.' return time else hh += 12 and return time
```

```
/* */
sub deltaDates
This subroutine determines the delta time between two dates
Algorithm
        Determine the greater date
        subtract from the greater date the lesser date
                subtract seconds and make necessary day corrections
                subtract days and make any necessarey year corrections
                subtract years
        return delta time

sub dateDelta
This subroutine computes the new date given the original date and delta time
Algorithm
        add delta time to date making corrections as necessary
        return new date
Calendar - c


/* */
int Gregorian_2_Julian(int month, int day, int year)

/* */
date Julian_2_Gregorian(int day, int year)


struct date
        int month
        int day
        int year
end struct

date Julian_2_Hobbit(int day, int year)
int Hobbit_2_Julian(int month, int day, int year)

int day_of_week_Gregorian(date)
int day_of_week_Hobbit(date)
int day_of_week_Julian(date)

Appointment_by_day()
Appointment_by_date()

Display_week
Display_day

Boolean isaLeapYear(int year)

*/
# calendar routines in C




    edate   itime,etime;

enum Gdow {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
enum GMon {January, February, March, April, May, June, July, August, September, October, Novembe…
enum Gry {31,28,31,30,31,30,31,31,30,31,30,31}
enum Gly {31,29,31,30,31,30,31,31,30,31,30,31}

}

int time2zulu (edate itime, edate ftime&);
........1.........2.........3.........4.........5.........6.........7.........8
/* ---------------------------------------------------------------------------- */
```

```
/* Convert a time from a local time zone to Greenwich Mean Time (Zulu)        */
/* v1.0 eab 12/2000                                                           */
/* Algorithm:                                                                 */
/*     First set the max number of days in the year by using isleap           */
/*     Now add the seconds for time zone correction to current seconds        */
/*     Now we do a series of checks to see if we changed to a different day   */
/*          If so we update the day and seconds, then see if that changes the */
/*          year and if so update the day and year accordingly                */
/* ------------------------------------------------------------------------- */

int zulu2time (edate itime, float newzone, edate ftime&);
/* ------------------------------------------------------------------------- */
/* Convert a Zulu time to anothor local time zone                             */
/* v1.0 eab 12/2000                                                           */
/* Algorithm:                                                                 */
/*     First set the max number of days in the year by using isleap           */
/*     Now subtract the seconds for time zone correction to current seconds   */
/*     Now we do a series of checks to see if we changed to a different day   */
/*          If so we update the day and seconds, then see if that changes the */
/*          year and if so update the day and year accordingly                */
/* ------------------------------------------------------------------------- */

int Gregorian2Julian (gdate itime, edate ftime&);
/* ------------------------------------------------------------------------- */
/* Convert a Gregorian date to a Julian date                                  */
/* v1.0 eab 12/2000                                                           */
/* Algorithm:                                                                 */
/*      First determine if the year is a leap year and set days/month array   */
/*                                                                            */

gdate Julian2Gregprium (edate itime)
{
}

edate Shire2Julian (sdate itime)
{
}

sdate Julian2Shire (edate itime)
{
}

string ShireDayofWeek()
string GregorianDayofWeek()

Boolean isleap(int year);
/* ------------------------------------------------------------------------- */
/* The algorithm used here is simplistic, refinement will be done later       */
/* Algorith:                                                                  */
/*     First return False for all years not divisible by 4                    */
/*           All years left are now evenly divisible by 4 so maybe leap year */
/*     Second return True for all years not divisible by 100                  */
/*           left is just the century years                                   */
/*     Return true if divisible by 400 otherwise return false                 */
/* ------------------------------------------------------------------------- */

float secs2time()
float time2secs()
nn 12convert24
nn 24convert12
nn deltaDates(edate itime, edate ftyme)
Functions
    FromGregorian
    ToGregorian
    GregorianDayofWeek
```

```
     FromShire
     ToShire
     ShireDayofWeek

     SecsTo24
     SecsTo12
     SecsTo10
     SecsFrom24
     SecsFrom12
     SecsFrom10

     IsLeap
     DeltaDates
     Functions
     FromGregorian
     ToGregorian
     GregorianDayofWeek
     FromShire
     ToShire
     ShireDayofWeek

     SecsTo24
     SecsTo12
     SecsTo10
     SecsFrom24
     SecsFrom12
     SecsFrom10

     IsLeap
     DeltaDates
/*****************************************************************************/
/*******************************   end   ************************************/
/*****************************************************************************/
A program for time and calendars

Time:
        24hour clock standard
        24hour world clock - AngularTime

Calendars
        Gregorian
        Shire
        Julian

Time:
        Internal time is seconds from midnight at GMT (0-86,400)
        Functions:
                Internal to standard 24 hour clock GMT (0-24,0-60,0-60)
                        secsrm = current seconds from midnight GMT
                        hours = trunc(secsrm/3600)
                        secrm = secsrm - (hours * 3600)
                        minutes = trunc(secsrm/60)
                        seconds = secsrm - (minutes * 60)
                GMT to local time
                        Hours = Hours + TimeZone  (+12 to -12)
                        if hours > 24 then day++ && hours=hours-24
                        if hours < 00 then day-- && hours=hours+24
                Internal to world clock (0-360,0-60,0-60)
                        secsrm = current seconds from midnight GMT
                        degrees = trunc(secsrm/240)
                        secrm = secsrm - (degrees * 240)
                        minutes = trunc(secsrm/60)
                        seconds = secsrm - (minutes * 60)
```

Calendars

Internal date is days from start of Gregorian year GMT

Functions:

Gregorian

Shire

Julian

```
==========================================================================================
/*****************************************************************************/
/* Etyme.m                                                                   */
/* ------------------------------------------------------------------------- */
/* Version: 1.0                                                              */
/*****************************************************************************/

/*****************************************************************************/
/********************                #includes                ********************/
/*****************************************************************************/
import 'Etyme.h'

@implementation
/*****************************************************************************/
/********************      the methods themselves          ********************/
/*****************************************************************************/
+ (NSOpenGLPixelFormat*) basicPixelFormat
{
        static NSOpenGLPixelFormatAttribute attributes[]=
                {
                        NSOpenGLPFAWindow,
                        NSOpenGLPFADoubleBuffer,
                        NSOpenGLFPADepthSize, (NSOpenGLPixelFormatAttribute)16,
                        (NSOpenGLPixelFormatAttribute)nil
                };
        return ([[[NSOpenGLLPixelFormat alloc] initWithAttributes:attributes] autorelease]);
}
/* ********** ********** ********** ********** ********** ********** */
- (void) resizeGL
{
}
/* ********** ********** ********** ********** ********** ********** */
- (void)drawRect:(NSREct)rect
{

}

/* ********** ********** ********** ********** ********** ********** */
/* Convert etyme to seconds                                           */
int etime2seconds(etyme itime, float secs)
{
        int errNum = 0;
        secs = itime.seconds + (60 * itime.minutes) + (3600 * itime.hours);
        return(errNum)
}

/* ********** ********** ********** ********** ********** ********** */
/* Seconds in day to 24 hour Earth time GMT                           */
int seconds2etime(float seconds, etyme theTime)
{
        int errNum = 0;

        if (seconds < 0) return(-1);    // negative time in date
        if (seconds >= 86400) return (1);  // more seconds than allowed in Earth day

        theTime.hour = (int)(seconds / 3600);
        seconds -= (theTime.hours * 3600);
        theTime.minutes = (int)(seconds / 60);
```

```
        theTime.seconds = seconds - (theTime.minutes * 60);
        return(errNum);
}

/* ********** ********** ********** ********** ********** ********** */
/* Convert angular time to seconds                                    */
int atime2seconds(atyme itime, float seconds)
{
        int errNum = 0;

        seconds = (itime.degrees * 240) + (itime.arc_minutes * 4) + (itime.arc_seconds / 15.0));
        return(errNum)
}

/* ********** ********** ********** ********** ********** ********** */
/* Convert seconds to angular time                                    */
int seconds2atime(float seconds, atyme theTime)
{
        int     errNum = 0;

        if (seconds < 0) return(-1);    // negative time in date
        if (seconds >= 86400) return (1);  // more seconds than allowed in Earth day

        theTime.degrees = (int)(seconds / 360);
        seconds -= (theTime.degrees * 360);
        theTime.arc_minutes = (int)(seconds / 60);
        theTime.arc_seconds = seconds - (theTime.arc_minutes * 60);
        return(errNum);
}

/* ********** ********** ********** ********** ********** ********** */
/* Convert a standard 24 hour Earth local time to zulu time           */
int local2zulu (etyme itime, float timeZone etyme ftime&)
{
        int errNum = 0;

        ftime.hour = itime + timeZone;
        return (errNum)
}

/* ********** ********** ********** ********** ********** ********** */
/* Convert a standard 24 hour Earth zulu time to local time           */
int zulu2local (etyme itime, float timeZone etyme ftime&)
{
        int errNum = 0;

        ftime.hour = itime - timeZone;
        return (errNum);
}

/* ********** ********** ********** ********** ********** ********** */
/* convert a zulu Earth time to a local Earth time zone */
int local2zulu (edate itime, float newzone, edate ftime&)
{
     edate    tyme;
     int      maxdays;
     int      errNum = 0;
     float    secs_per_day=24.0*60.0*60.0;

     isleap(itime.year) ? maxdays = 366 : maxdays = 365;
     tyme.secs = itime.secs + (newzone * 3600)
     if (tyme.secs > (secs_per_day - 1)
       {
         tyme.secs -= secs_per_day;
         tyme.days += 1;
```

```
        }
      else
      if (tyme.secs < 0)  /* before midnight */
        {
           tyme.secs += secs_per_day;
           tyme.days -= 1;
        }
      if (tyme.days > maxdays)   /* next year */
        {
           tyme.days -= maxdays;
           tyme.year += 1;
        }
      else
      if ( tyme.days < 1 )   /* last year */
        {
           tyme.days += maxdays;
           tyme.year -= 1;
        }
      tyme.zone = 0;
      ftime = tyme;
      return (errNum);
}

/* ********** ********** ********** ********** ********** ********** */
/* convert a zulu Earth time to a local Earth time zone */
int zulu2local (edate itime, float newzone, edate ftime&)
{
      edate    tyme;
      int      maxdays;
      int      errNum =0;
      float    secs_per_day=24.0*60.0*60.0;

      isleap(itime.year) ? maxdays = 366 : maxdays = 365;
      tyme.secs = itime.secs - (newzone * 3600)
      if (tyme.secs > (secs_per_day - 1)
        {
           tyme.secs -= secs_per_day;
           tyme.days += 1;
        }
      else
      if (tyme.secs < 0)  /* before midnight */
        {
           tyme.secs += secs_per_day;
           tyme.days -= 1;
        }
      if (tyme.days > maxdays)   /* next year */
        {
           tyme.days -= maxdays;
           tyme.year += 1;
        }
      else
      if ( tyme.days < 1 )   /* last year */
        {
           tyme.days += maxdays;
           tyme.year -= 1;
        }
      tyme.zone = newzone;
      ftyme = tyme;
      return (errNum);
}

/* ********** ********** ********** ********** ********** ********** */
/* Is this a Earth Gregorian calendar leap year                       */
Boolean isleap(int year)
{
```

```
        Boolean isaLeapYear;

         if (year mod 4) <> 0 then
                isaLeapYear = FALSE;
        else
                isaLeapYear = TRUE;
        If (((year mod 100) == 0) and ((year mod 400) <> 0)) then
                isaLeapYear = FALSE;
        return (isaLeapYear);
}

/* ********** ********** ********** ********** ********** ********** */
/* Convert a Gregorian Calendar date and time to internal date&time  */
int gregorian2internal(gdates itime, edaet ftime)
{
        int     errNum = 0;
        int maxdays;
        struct etyme theTime;

        isleap(itime.year) ? maxdays = 366 : maxdays = 365;
        if (itime.zone != 0)
        {
                errNum = local2zulu(itime.etyme, itime.zone, theTyme);
                if (theTyme.hour > 23)
                {
                        itime.day += 1;
                        theTyme.hour -= 24;
                }
                if (theTyme.hour < 0 )
                {
                        itime.day -= 1;
                        theTyme.hour += 24;
                }
        }
        errNum = etime2seconds(theTyme, ftime.seconds);
        ftime.year = itime.year;
        switch (itime.month)
        {
                case  1: // January - 31
                                ftime.days = itime.day;
                                if (itime.day == 0 )
                                {
                                        isleap((itime.year - 1)) ? ftime.days = 366 : ftime.days…
                                        ftime.year -= 1;
                                }
                                break;
                case  2: // February - 28 ? 29
                                ftime.days = itime.day + 31;
                                break;
                case  3: // March - 31
                                isleap(itime.year) ? ftime.days = 59 + itime.day : ftime.days = …
                                break;
                case  4: // April - 30
                                isleap(itime.year) ? ftime.days = 90 + itime.day : ftime.days = …
                                break;
                case  5: // May - 31
                                isleap(itime.year) ? ftime.days = 120 + itime.day : ftime.days =…
                                break;
                case  6: // June - 30
                                isleap(itime.year) ? ftime.days = 151 + itime.day : ftime.days =…
                                break;
                case  7: // July - 31
                                isleap(itime.year) ? ftime.days = 181 + itime.day : ftime.days =…
                                break;
                case  8: // August - 31
```

```
                        isleap(itime.year) ? ftime.days = 212 + itime.day : ftime.days =…
                        break;
            case  9: // September - 30
                        isleap(itime.year) ? ftime.days = 243 + itime.day : ftime.days =…
                        break;
            case 10: // October - 31
                        isleap(itime.year) ? ftime.days = 273 + itime.day : ftime.days =…
                        break;
            case 11: // November - 30
                        isleap(itime.year) ? ftime.days = 304 + itime.day : ftime.days =…
                        break;
            case 12: // December - 31
                        isleap(itime.year) ? ftime.days = 334 + itime.day : ftime.days =…
                        if (ftime.days > maxdays)
                        {
                                ftime.year += 1;
                                ftime.days = 1;
                        }
                        break;
        }
        return (errNum);
}

/* ********** ********** ********** ********** ********** ********** */
/* Convert an internal date and time to Gregorian date and time      */
/* Note that if year == 1582 10/5==10/15
{
}

/* ********** ********** ********** ********** ********** ********** */
int angular2zulu (edate itime, edate ftime)
/* ********** ********** ********** ********** ********** ********** */
int zulu2angular (edate itime, edate ftime)
/* ********** ********** ********** ********** ********** ********** */
int Shire2internal(gdates itime, edaet ftime)int gregorian2internal(gdates itime, edaet ftime)
{
        int     errNum = 0;
        int maxdays;
        struct etyme theTime;

        isleap(itime.year) ? maxdays = 366 : maxdays = 365;
        if (itime.zone != 0)
        {
                errNum = local2zulu(itime.etyme, itime.zone, theTyme);
                if (theTyme.hour > 23)
                {
                        itime.day += 1;
                        theTyme.hour -= 24;
                }
                if (theTyme.hour < 0 )
                {
                        itime.day -= 1;
                        theTyme.hour += 24;
                }
        }
        errNum = etime2seconds(theTyme, ftime.seconds);
        ftime.year = itime.year;
        switch (itime.month)
        {
                case  0: // Days outside of months
                                if (itime.day == 1) ftime.days = 1;
                case  1: // Yule
                                ftime.days = itime.day + 1;
                                if (itime.day < 0 )
                                {
```

```
                                isleap((itime.year - 1)) ? ftime.days = 366 : ftime.days…
                                ftime.year -= 1;
                        }
                        break;
                case  2: // Solmath
                        ftime.days = itime.day + 31;
                        break;
                case  3: // Rethe
                        ftime.days = itime.day + 61;
                        break;
                case  4: // Astron
                        ftime.days = itime.day + 91;
                        break;
                case  5: // May - 31
                        ftime.days = itime.day + 121;
                        break;
                case  6: // June - 30
                        ftime.days = itime.day + 151;
                        break;
                case  7: // July - 31
                        isleap(itime.year) ? ftime.days = 181 + itime.day : ftime.days =…
                        break;
                case  8: // August - 31
                        isleap(itime.year) ? ftime.days = 212 + itime.day : ftime.days =…
                        break;
                case  9: // September - 30
                        isleap(itime.year) ? ftime.days = 243 + itime.day : ftime.days =…
                        break;
                case 10: // October - 31
                        isleap(itime.year) ? ftime.days = 273 + itime.day : ftime.days =…
                        break;
                case 11: // November - 30
                        isleap(itime.year) ? ftime.days = 304 + itime.day : ftime.days =…
                        break;
                case 12: // December - 31
                        isleap(itime.year) ? ftime.days = 334 + itime.day : ftime.days =…
                        if (ftime.days > maxdays)
                        {
                                ftime.year += 1;
                                ftime.days = 1;
                        }
                        break;
        }
        return (errNum);
}

/* ********** ********** ********** ********** ********** ********** */
int internal2Shire()
/* ********** ********** ********** ********** ********** ********** */
int dectime2internal
/* ********** ********** ********** ********** ********** ********** */
int Fixed2internal
/* ********** ********** ********** ********** ********** ********** */
int world2internal
/* ********** ********** ********** ********** ********** ********** */

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

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

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

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

@end
```

```
/*****************************************************************************/
/********************                end                   ********************/
/*****************************************************************************/

# calendar routines in C?C++




int Gregorian2Julian (gdate itime, edate tyme&)
/* ------------------------------------------------------------------------- */
/* Convert a Gregorian date to a Julian date                                 */
/* v1.0 eab 12/2000                                                          */
/* Algorithm:                                                                */
/*      First determine if the year is a leap year and set days/month array  */
/*                                                                           */
{                                                                            */
         int   errNum = 0;
     short maxdays;

     short leapdays [31,29,31,30,31,30,31,31,30,30,30,31];
     short regdays  [31,29,31,30,31,30,31,31,30,30,30,31];

     isleap(itime.year) ? dayinmonth = leapdays : regdays ;
     tyme.secs = itime.secs + (60.0 * itime.hours);
     tyme.year = itime.year;
     tyme.zone = itime.zone;
     if ( itime.month == 0 )
       {
          tyme.days = itime.day
       }
     else
       {
          tyme.days = 0;
          for (i=1 : i < (itime.month-1) : i++)
             {
               tyme.days += dayinmonth[i-1];
             }
          tyme.days += itime.day;
     return(errNum);
}

int Julian2Gregoriam (edate itime, gdate ftyme&)
{
}

edate Shire2Julian (sdate itime)
{
}

sdate Julian2Shire (edate itime)
{
}

string ShireDayofWeek()
{
}

string GregorianDayofWeek()
{
}

float secs2time()
float time2secs()
nn 12convert24
```

```
nn 24convert12
nn deltaDates(edate itime, edate ftyme)
==========================================================================================…
#Chinese Zodiac animals

year = int(input("Enter a year: "))
zodiacYear = year % 12
if zodiacYear == 0:
    print("monkey")
elif zodiacYear == 1:
    print("rooster")
elif zodiacYear == 2:
    print("dog")
elif zodiacYear == 3:
    print("pig")
elif zodiacYear == 4:
    print("rat")
elif zodiacYear == 5:
    print("ox")
elif zodiacYear == 6:
    print("tiger")
elif zodiacYear == 7:
    print("rabbit")
elif zodiacYear == 8:
    print("dragon")
elif zodiacYear == 9:
    print("snake")
elif zodiacYear == 10:
    print("horse")
else:
    print("sheep")
==========================================================================================…
#! perl -w

=pod
=head1 Header

     Name:    calen.pl
  Version:    1.0
     Date:    12/2001
   Author:    Edward Birdsall
  Language:   Perl

=head1 Introduction

This file contains a set of calendar routines used in various programs using the
Perl language.  I find that they come in handy so I develop them in the various
computer languages I wish to use for programming.  The Julian and Gregorian
calendars are the most commonly used and I have added Shire and Alt calendars
for variety.  I may add other calendars later as I get time and interest.

=head1 Notes

#23456789-123456789-123456789-123456789-123456789-123456789-123456789-123456789-
#-----------------------------------------------------------------------------
=head2 internal time formats

These functions use Julian GMT time in the form ($year, $day, $secs) with
seconds in real form.  Time zones are in +/- seconds from 0 to 43200.

=head2 perl time functions of note

Greenwich Time functions

 ($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst) = gmtime(time);
```

```
  timegm function use Time::Local;
  $time = timegm($sec, $min, $hour, $mday, $mon, $year);

 Localtime functions
  ($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst) = localtime(time);
  timelocal function use Time::Local;
  $time = timelocal($sec, $min, $hour, $mday, $mon, $year);

 The year 1900 == 0 and most of the variables are 0 indexed vice 1 indexed for
 start

 = head2 zone2zone

 This function converts the time from one time zone to another.

 =cut

 sub zone2zone {
         my $secs = $time_zone + $secs;
      if (my $secs > 86400) {
           my $secs -= 86400;
           my $day  += 1;
      }
      elseif (my $secs < 0) {
           my $secs += 86400;
           my $day  -= 1;
      }
      maxdays = 365 = is_leap(my $year);
      if (my $days > maxdays) {
           my $days -= maxdays;
           my $year += 1;
      }
      elseif (my $days < 1) {
           my $days += maxdays;
           my $year -= 1;
 }

 =head2 gregorian2julian
 sub gregorian2julian
 sub julian2gregorain
 sub gregoriandayofweek

 =head1 Shire calendar routines

 The Shire calendar found in J.R.Tolkein Ring series has twelve months of thirty
 days with four days not in a month and one or two not in a month or week.  Each
 year starts on 2Yule which is Sterrendei (equivalent to Saturday).  The year
 pattern is 2Yule, 1-30 Afteryule, 1-30 Solmath, 1-30 Rethe, 1-30 Astron, 1-30
 Thrimidge, 1-30 Forelithe, 1Lithe, [Midyear's Day, (Overlithe)], 2Lithe, 1-30
 Afterlithe, 1-30 Wedmath, 1-30 Halimath, 1-30 Winterfilth, 1-30 Blotmath,
 1-30 Foreyule, 1Yule.  Overlithe added for leap years and Midyear's Day are not
 part of any week.

 Days of the week are; Sterrendei, Sunnendei, Momendei, Trewesdei,Hevenesdei,
 Meresdei, and Highdei.

 =head2 shire2julian

 This function converts the Shire calendar information back to Julian.

 =cut

 sub shire2julian

 =head2 julian2shire
```

```
123456789-123456789-123456789-123456789-123456789-123456789-123456789-
This converts a time from Julian to the Shire calendar.

=cut
sub julian2shire {
}

sub shire_dow {
}

=head1 Alt calendar

This calendar was proposed a while ago and I don't remmember all the details.
It has thirteen months of twenty eight days with MidDay between the fouteenth and
fifteenth day of the seventh month.  MidDay and LeapDay which is added on leap
years are not part of the week similar to the Shire Calendar.

=head1 Generic calendar routines

These routines apply to all calendars herein and are lumped here.

=head2 isleap

This subroutine determines if the year in question is a leap year.  The rule
applied is that leap years are every fourth year except century years unless
it is every fouth century.

=cut
sub isleap{
}

=head2 time2secs
=cut
sub time2secs {
}
sub secs2time {
}
sub 12convert24 {
}
sub 24convert12 {
}
sub deltadates {
}
sub datedelta {
}

==========================================================================================
/*****************************************************************************************
/* Calendar.h                                                                            */
/*-------------------------------------------------------------------------------------*/
/* current v1.00 July 2004                                                             */
/*=====================================================================================*/
/*   Purpose                                                                           */
/*       This program is        */                                                      */
/*  */
/*  */
/* ------------------------------------------------------------------------------------ */
/*   Platform:  all                                                                    */
/*-------------------------------------------------------------------------------------*/
/*   History                                                                           */
/*   -------                                                                           */
/*****************************************************************************************/

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

```
/*************************     #includes     ********************************/
/*****************************************************************************/
#include ".h"

/*****************************************************************************/
/*************************   typedef, define enum   **************************/
/*****************************************************************************/

#define BackSize         50         /* size of backtrack buffer originally 20 */

/*****************************************************************************/
/*************************   function prototypes    **************************/
/*****************************************************************************/

/* Public - used by other modules */
/*--------------------------------*/

/* Private - for use by this module only */
/*---------------------------------------*/

/*****************************************************************************/
/*************************   function descriptions    *************************/
/*****************************************************************************/

/*****************************************************************************/
/*************************     Public functions       *************************/
/*****************************************************************************/

/*****************************************************************************/
/* module                                                                    */
/* ------------------------------------------------------------------------- */
/* Purpose:                                                                  */
/* Version:                                                                  */
/* Author:                                                                   */
/* Date:                                                                     */
/* ------------------------------------------------------------------------- */
/* Input:                                                                    */
/* Returns:                                                                  */
/* Other:                                                                    */
/* Functions Used (with module/library):                                     */
/*       Function                - module or library                         */
/* ------------------------------------------------------------------------- */
/* Operation:                                                                */
/* ------------------------------------------------------------------------- */
/* Version history:                                                          */
/* Version - Date    - Author        - Notes                                 */
/*****************************************************************************/

/*****************************************************************************/
/*************************    Private functions       *************************/
/*****************************************************************************/


/*****************************************************************************/
/* module                                                                    */
/* ------------------------------------------------------------------------- */
/* Purpose:                                                                  */
/* Version:                                                                  */
/* Author:                                                                   */
/* Date:                                                                     */
/* ------------------------------------------------------------------------- */
/* Input:                                                                    */
/* Returns:                                                                  */
                                                                             */
```

```
/* Other:
/* Functions Used (with module/library):                                  */
/*       Function                - module or library                      */
/* ------------------------------------------------------------------------ */
/* Operation:                                                               */
/* ------------------------------------------------------------------------ */
/* Version history:                                                         */
/* Version - Date    - Author        - Notes                                */
/*************************************************************************/
```

Calendar Program

This program has a number of different functions

1) Convert between any two calendar systems that I know
2) Display a calendar period (month, week, day) of a desired system
3) Appointments
4) Compute delta date/time

Calendar systems

- Julian
- Gregorian
- Shire
- 'Scientific'-a
- Mars

Structures

```
jdate - secs from midnight
        day from year start
        year
        time zone delta hour from Zulu
```

```
/* ===== * ===== * ===== * ===== ****** ===== * ===== * ===== * ===== * ===== */
/* Calendar systems
/* ----------------                                                         */
/* Julian                                                                   */
/*       Year - Day - Time                                                  */
/*                                                                          */
/*     Year - could be Christian or some other - pick is Christian          */
/*     Day - First day of the year is 1 and then sequentially numbered till */
/*           end of year                                                    */
/*     Time - Earth 24 hour day at Zulu (Greenwich Mean Time)               */
/*            24 hours/day, 60 minutes/hour, 60 seconds/minute              */
/* ----- -----                                                              */
/* Gregorian                                                                */
/*       Year - Month - Day of Month - Time - time zone - Day of Week       */
/*              This calendar was started in year      A.D. with a date shift */
/*                   from the existing calendar so need to put that in the  */
/*                   algorithm.                                             */
/*                                                                          */
/*       Year - Christian calendar wih A.D. and B.C.                        */
/*       Month - 12 months of varying days, second month changes number of  */
/*               days if leap year                                          */
/*            January-31, February-28?29, March-31, April-30, May-31, June-30 */
/*            July-31, August-31, September-30, October-31, November-30,    */
/*            December-31                                                   */
/*       Day of Month - see above for days in each month                    */
/*       Time - Earth 24 hour time                                          */
/*       Time Zone - ranges from -12 to +12 with 0 being Zulu               */
```

```
/*      Day of Week - 7 days per week,  each year starts on a different day  */
/*           Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday  */
/* ----- -----                                                              */
/* Shire                                                                    */
/*      Year - Month - Day of Month - Time - time zone - Day of Week        */
/*                                                                          */
/*      Year - ? */                                                         */
/*      Month - 12 months of 30 days - 5 days not part of any month or week  */
/*           Afteryule, Solmath, Rethe, Astron, Thrimidge, Forelithe,       */
/*           Afterlithe, Wedmath, Halimath, Winterfilth, Blotmath, Foreyule  */
/*      Day of month - 1 to 30 with a few days not part of any month        */
/*      Time - Earth 24 hour time                                           */
/*      Time Zone - ranges from -12 to +12 with 0 being Zulu                */
/*      Day of Week - 7 days per week, each year starts on Sterrendei       */
/*           Sterrendei, Sunnendei, Momendei, Trewesdei, Hevenesdei,        */
/*           Meresdei, Highdei                                              */
/*      Year's progression                                                  */
/*           2Yule, Afteryule, Solmath, Rethe, Astron, Thrimidge, Forelithe, */
/*           1Lithe,                                                        */
/*           Midyear's Day, (OverLithe), - these two are not part of the week*/
/*           2Lithe, Afterlithe, Wedmath, Halimath, Winterfilth, Blotmath,  */
/*           Foreyule, 1Yule                                                */
/* ----- -----                                                              */
/* Scientific                                                               */
/*      Year - Month - Day of Month - Time - time zone - Day of Week        */
/*                                                                          */
/*      Year - ?                                                            */
/*      Month - 13 months                                                   */
/*      Day of month - 1 through 28 with MidDay and LeapDay not part of month*/
/*      Time - Earth 24 hour time                                           */
/*      Time Zone - ranges from -12 to +12 with 0 being Zulu                */
/*      Day of Week - 7 days per week,  each year starts on Sunday          */
/*           Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday  */
/*           MidDay and LeapDay not part of any week                        */
/*           MidDay is in the 7th month between the 14th and 15th           */
/*           LeapDay is with MidDay on leap years                           */
/* ----- -----                                                              */
/* Mars1 */                                                                  */
/*  MDay = 24 hours, 39 minutes, 35 seconds of ETime */
/*  MYear =  668.62... MDays    = 687 EDays  */

Structures
    jdate - secs from midnight
            day from year start
            year
            time zone delta hour from Zulu



Years divisible by 4 a leap year except divisible by 100 unless divisible by 400

*****************************************************************************
Internal date:time format is Julian GMT -  year:day:sssss.sss
time zones internal sssss
/* */
sub time2zulu { }
/* This subroutine converts a zone julian date:time to zulu given original date:time and time zo…
Algorithm:
        add time to original
        if time > 86400 then subtract 86400 from time and increment date
        if time < 0000 then add 2400 to time and decrement date
        if day > year's end then date=1 and increment year
        if day < 1 then day=1 and decrement year
/* */
```

```
sub zulu2time { }
This subroutine converts a zulu date:time to zone given zulu date:time and time zone
Algorithm:
        subtract time from zulu
        if time > 2400 then subtract 2400 from time and increment date
        if time < 0000 then add 2400 to time and decrement date
        if day > year's end then date=1 and increment year
        if day < 1 then day=1 and decrement year
/* */
int Gregorian2Julian();
/* This subroutine converts a Gregorian date to a Julian date                */
Algorithm:
        Determine if leap year and use appropriate month set
        determine integer month
        return day = day of month plus sum of previous months

sub Julian2Gregorian { }
This subroutine converts a Julian date to a Gregorian date
Algorithm:
        Determine if leap year and use appropriate month set
        While day !inMonth increment month
        Return Gregorian Date
/* */
sub Shire2Julian { }
This subroutine converts a Shire date to a Julian date
Algorithm:
        If special day then set value and return day
        Determine if leap year and use appropriate year set
        determine integer month
        return day = day of month plus sum of previous months
/* */
sub Julian2Shire { }
This subroutine converts a Julian date to a Shire date
Algorithm:
        Determine if leap year and use appropriate year set
        If special day then return Shire date by table
        While day !inMonth increment Month
        return Shire date
/* */
sub ShireDayofWeek { }
This subroutine returns the number of the day of week 0:7 given a Shire date
Algorithm:
/* */
sub GregorianDayofWeek { }
This subroutine returns the number of the day of week 1:7 given a Gregorian Date
Algorithm:
/* */
sub isLeap { }
This returns True if LeapYear otherwise False
Algorithm:
        if year mod 4  0 then return False
        if year mod 4 = 0 and year mod 400 = 0 return True
        if year mod 4 = 0 and year mod 100  0 return True else False
/* */
sub time2secs { }
This subroutine converts 24 hour time to seconds from midnight
Algorithm:
/* */
sub secs2time ( )
This subroutine converts seconds from midnight to 24 hour time
Algorithm:
        ss.sss = secs mod 60
        hour = trunc(secs - ss.sss)/3600
        minutes = ((secs - ss.sss) - (hour*3600))/60
/* */
```

```
sub 12convert24
This sub converts a 24 hour time to 12 hour time
Algorithm:
        if hh < 13 return time + 'a.m.'
                else hh -= 12 and return time + 'p.m.'
/* */
sub 24convert12
This sub converts a 12 hour time to 24 hour time
Algorithm:
        if 'a.m.' return time else hh += 12 and return time
/* */
sub deltaDates
This subroutine determines the delta time between two dates
Algorithm
        Determine the greater date
        subtract from the greater date the lesser date
                subtract seconds and make necessary day corrections
                subtract days and make any necessarey year corrections
                subtract years
        return delta time

sub dateDelta
This subroutine computes the new date given the original date and delta time
Algorithm
        add delta time to date making corrections as necessary
        return new date
Calendar - c


/* */
int Gregorian_2_Julian(int month, int day, int year)

/* */
date Julian_2_Gregorian(int day, int year)


struct date
        int month
        int day
        int year
end struct

date Julian_2_Hobbit(int day, int year)
int Hobbit_2_Julian(int month, int day, int year)

int day_of_week_Gregorian(date)
int day_of_week_Hobbit(date)
int day_of_week_Julian(date)

Appointment_by_day()
Appointment_by_date()

Display_week
Display_day

Boolean isaLeapYear(int year)

*/
# calendar routines in C



struct edate {
    float   secs;    /* seconds from midnight  0->86399 */
    short   days;    /* days in year 1->365 or 366      */
```

```
    int     year;    /* year                                  */
    float   zone;    /* time zone -12->+12                    */
} edate;

struct gdate {
    float   secs;
    short   hour;
    short   day;
    int     year;
    float   zone;
} gdate;

    edate   itime,etime;

enum Gdow {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
enum GMon {January, February, March, April, May, June, July, August, September, October, Novembe…
enum Gry {31,28,31,30,31,30,31,31,30,31,30,31}
enum Gly {31,29,31,30,31,30,31,31,30,31,30,31}

}

int time2zulu (edate itime, edate ftime&);
........1........2........3........4........5........6........7........8
/* ---------------------------------------------------------------------- */
/* Convert a time from a local time zone to Greenwich Mean Time (Zulu)    */
/* v1.0 eab 12/2000                                                       */
/* Algorithm:                                                             */
/*     First set the max number of days in the year by using isleap       */
/*     Now add the seconds for time zone correction to current seconds    */
/*     Now we do a series of checks to see if we changed to a different day  */
/*          If so we update the day and seconds, then see if that changes the */
/*          year and if so update the day and year accordingly            */
/* ---------------------------------------------------------------------- */

int zulu2time (edate itime, float newzone, edate ftime&);
/* ---------------------------------------------------------------------- */
/* Convert a Zulu time to anothor local time zone                         */
/* v1.0 eab 12/2000                                                       */
/* Algorithm:                                                             */
/*     First set the max number of days in the year by using isleap       */
/*     Now subtract the seconds for time zone correction to current seconds  */
/*     Now we do a series of checks to see if we changed to a different day  */
/*          If so we update the day and seconds, then see if that changes the */
/*          year and if so update the day and year accordingly            */
/* ---------------------------------------------------------------------- */

int Gregorian2Julian (gdate itime, edate ftime&);
/* ---------------------------------------------------------------------- */
/* Convert a Gregorian date to a Julian date                              */
/* v1.0 eab 12/2000                                                       */
/* Algorithm:                                                             */
/*      First determine if the year is a leap year and set days/month array  */
/*                                                                        */

gdate Julian2Gregprium (edate itime)
{
}

edate Shire2Julian (sdate itime)
{
}

sdate Julian2Shire (edate itime)
{
}
```

```
string ShireDayofWeek()
string GregorianDayofWeek()

Boolean isleap(int year);
/* ------------------------------------------------------------------------- */
/* The algorithm used here is simplistic, refinement will be done later      */
/* Algorith:                                                                 */
/*      First return False for all years not divisible by 4                  */
/*            All years left are now evenly divisible by 4 so maybe leap year */
/*      Second return True for all years not divisible by 100                */
/*            left is just the century years                                 */
/*      Return true if divisible by 400 otherwise return false               */
/* ------------------------------------------------------------------------- */

float secs2time()
float time2secs()
nn 12convert24
nn 24convert12
nn deltaDates(edate itime, edate ftyme)
Functions
    FromGregorian
    ToGregorian
    GregorianDayofWeek
    FromShire
    ToShire
    ShireDayofWeek

    SecsTo24
    SecsTo12
    SecsTo10
    SecsFrom24
    SecsFrom12
    SecsFrom10

    IsLeap
    DeltaDates
    Functions
    FromGregorian
    ToGregorian
    GregorianDayofWeek
    FromShire
    ToShire
    ShireDayofWeek

    SecsTo24
    SecsTo12
    SecsTo10
    SecsFrom24
    SecsFrom12
    SecsFrom10

    IsLeap
    DeltaDates
/*****************************************************************************/
/*******************************   end   ************************************/
/*****************************************************************************/
=============================================================================================================

/*****************************************************************************/
/* Main.c                                                                    */
/* ------------------------------------------------------------------------- */
/* Version:                                                                  */
/*****************************************************************************/
```

```
/*****************************************************************************/
/********************                #includes                ********************/
/*****************************************************************************/

/*****************************************************************************/
/********************     the functions themselves         ********************/
/*****************************************************************************/

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

/*****************************************************************************/
/********************                end                      ********************/
/*****************************************************************************/
# calendar routines in C

int main( int argc, char **argv)
{
     /* a few variables defined here */
     BOOL   done  FALSE;

     /* initialize the machine */

     /* initialize graphics */

     /* opening screen */

     /* where do we go from here */

     /* Main Loop */
struct edate {
     float   secs;    /* seconds from midnight  0->86399 */
     short   days;    /* days in year 1->365 or 366      */
     int     year;    /* year                            */
     float   zone;    /* time zone -12->+12              */
} edate;

struct gdate {
     float   secs;
     short   hour;
     short   day;
     int     year;
     float   zone;
} gdate;

     edate   itime, etime;

     exit(0);
}

int time2zulu (edate itime, edate ftime&)   /* Convert a time to zulu time */
{
     edate    tyme;
     int      maxdays;
     int      errNum = 0;
     float    secs_per_day=24.0*60.0*60.0;

     isleap(itime.year) ? maxdays = 366 : maxdays = 365;
     tyme.secs = itime.secs + (itime.zone * 3600)
     if (tyme.secs > (secs_per_day - 1)
       {
          tyme.secs -= secs_per_day;
          tyme.days += 1;
       }
     else
```

```
     if (tyme.secs < 0)  /* before midnight */
       {
          tyme.secs += secs_per_day;
          tyme.days -= 1;
       }
     if (tyme.days > maxdays)   /* next year */
       {
          tyme.days -= maxdays;
          tyme.year += 1;
       }
     else
     if ( tyme.days < 1 )   /* last year */
       {
          tyme.days += maxdays;
          tyme.year -= 1;
       }
     tyme.zone = 0.0;
     ftime = tyme;
     return (errNum);
}

int zulu2time (edate itime, float newzone, edate ftime&)   /* convert a zulu time to a local tim…

     edate    tyme;
     int      maxdays;
     int      errNum =0;
     float    secs_per_day=24.0*60.0*60.0;

     isleap(itime.year) ? maxdays = 366 : maxdays = 365;
     tyme.secs = itime.secs - (newzone * 3600)
     if (tyme.secs > (secs_per_day - 1)
       {
          tyme.secs -= secs_per_day;
          tyme.days += 1;
       }
     else
     if (tyme.secs < 0)  /* before midnight */
       {
          tyme.secs += secs_per_day;
          tyme.days -= 1;
       }
     if (tyme.days > maxdays)   /* next year */
       {
          tyme.days -= maxdays;
          tyme.year += 1;
       }
     else
     if ( tyme.days < 1 )   /* last year */
       {
          tyme.days += maxdays;
          tyme.year -= 1;
       }
     tyme.zone = newzone;
     ftyme = tyme;
     return (errNum);
}

int Gregorian2Julian (gdate itime, edate tyme&)
/* ------------------------------------------------------------------------- */
/* Convert a Gregorian date to a Julian date                                 */
/* v1.0 eab 12/2000                                                          */
/* Algorithm:                                                                */
/*      First determine if the year is a leap year and set days/month array  */
/*                                                                           */
{                                                                            */
```

```
        int   errNum = 0;
      short maxdays;

      short leapdays [31,29,31,30,31,30,31,31,30,30,30,31];
      short regdays  [31,29,31,30,31,30,31,31,30,30,30,31];

      isleap(itime.year) ? dayinmonth = leapdays : regdays ;
      tyme.secs = itime.secs + (60.0 * itime.hours);
      tyme.year = itime.year;
      tyme.zone = itime.zone;
      if ( itime.month == 0 )
        {
           tyme.days = itime.day
        }
      else
        {
           tyme.days = 0;
           for (i=1 : i < (itime.month-1) : i++)
              {
                tyme.days += dayinmonth[i-1];
              }
           tyme.days += itime.day;
      return(errNum);
 }

 int Julian2Gregoriam (edate itime, gdate ftyme&)
 {
 }

 edate Shire2Julian (sdate itime)
 {
 }

 sdate Julian2Shire (edate itime)
 {
 }

 string ShireDayofWeek()
 string GregorianDayofWeek()

 Boolean isleap(int year)
 {
      if (year mod 4 != 0) return (False);
      if (year mod 100 != 0) return (True);
      if (year mod 400 == 0) return (True);
      return(False);
 }
         If (year mod 4) <> 0 then
                 isaLeapYear = FALSE;
         else
                 isaLeapYear = TRUE;
         If (((year mod 100) == 0) and ((year mod 400) <> 0)) then
                 isaLeapYear = FALSE;
         return isaLeapYear;

float secs2time()
float time2secs()
nn 12convert24
nn 24convert12
nn deltaDates(edate itime, edate ftyme)
==========================================================================================================...
```