SKYCALC USER'S MANUAL

John Thorstensen, Dept. Physics and Astronomy, Dartmouth College

0. Introduction – Why Bother? 2

1. The Interactive Almanac 2
  Overview 2
  1.1 Basic Use of the Program 4
    Starting up 4
    Specifying a date (`y`) and Getting an Almanac (`a`) 5
    Specifying RA and dec (`r` and `d`) and tabulating hourly airmass (`h`) . . . . 6 *ff*
    A Word about Hard Copy and Log Files 8
    Specifying time (`t`) and getting Instantaneous Circumstances with `=` . . . 9 *ff*
    Observability through a Season (`o`) 11
    Looking at Current Parameters with `l` 12
  1.2 Other Calculator-Mode Commands 13
    Quitting the program (`Q`) 13
    A Word about 'eXtra Goodies' (`x`) 14
    Setting the Time to Now – `T` 14
    Changing the site (`s`) 14
    UT time input and 'night dates' (`g` and `n`) 15
    Coordinate epoch (`e`) and batch precession (`xb`) 16
    Proper Motions (`p`) 16
    Coordinate conversions (`xc`) 17
    Julian Date calculations and setting (`xJ` and `xJ`) 17
    TDT – UT calculation (`xd`) 17
    Major planets (`m`) 18
    Setting to the Zenith (`xZ`) 18
    Object list handling – `xR, xl, xN,` and `xS` (`m`) 19
  1.3 Algorithms, Accuracy and Limitations 22
    Calendars and times 22
    Sun and Moon 23
    The Major Planets 25
    Geographical Limitations 26
    Precession 26
    Local Mean Sidereal Time 27
    Parallactic Angle 27
    Barycentric ('Heliocentric') Corrections 27
    Galactic and Ecliptic Coordinates 28
    Bugs and other problems . . . . . 28 *ff.*
    Programmer's Notes 29

2. A Nighttime Astronomical Calendar 30
  General description 30
  Times in the Calendar Program 32
  Running the Calendar 32
  More on TeX Output; Sample Output 34

3. Cautions Applying to Both Programs; Miscellany. 35

0. INTRODUCTION – WHY BOTHER?

You've just received your time assignment for Kitt Peak and you wonder whether the moon will interfere with your objects during those nights, which *weren't* the one's you asked for. Or, you're sitting at the telescope at 1 AM, wondering if you can squeeze in a 1-hour exposure before twilight at acceptable airmass on an object that's just rising now. Maybe you want to set your spectrograph slit to neutralize atmospheric dispersion. Maybe you want to precess a couple of objects' coordinates, or see what their galactic latitude is. Perhaps you want to spot-check the canned heliocentric corrections which IRAF has applied to all your data. Perhaps you just want to know how high the sun will be above the horizon at 4 PM in October, so you can see if it's safe to take a bike ride. Any competent astronomer armed with some reference materials and a calculator can answer these questions, but it takes time. Over the years I've done this sort of thing many times; I finally decided to encapsulate some utility routines of this sort into a couple of convenient, easy-to-use, portable packages.

This document describes two programs. The more powerful and interesting one is an interactive astronomical circumstances calculator. The other prints a 1-year nighttime *calendar* of phenomena for a single site; this will generally be run in background, to produce a table to be hung on the observatory wall or put in a notebook.

I wrote both these as standalone C-language programs. To maximize portability and ease of use I tried to make the user interface "as simple as possible, but no simpler" (to paraphrase Einstein). There are no graphics, no mouse-driven menus, or anything like that. You type stuff and the computer types stuff back. The commands are as terse as possible – single letters – so even hunt-and-peck typists should be able to use the programs efficiently. Throughout this document I've indicated things that you or the computer type with `this typeface`.

In some sense these programs and their documentation are a 'publication' for me, though not in a refereed journal. Accordingly, I'd like these to be disseminated as widely as possible in the community of professional astronomers. Please feel free to pass them along. If you would like your own copy, it can be obtained via anonymous ftp from `iraf.noao.edu`; it's in the `contrib` directory under the name `skycalc`. You'll need access to a UNIX machine to unpack the `tar` file, but the code itself should run on almost anything with a C compiler.

1. THE INTERACTIVE ALMANAC

Overview

This is designed to provide quick and easy access to astronomical quantities of interest to an observer at the telescope, and to ease the planning of (especially nighttime) observations.

This may seem unnecessary, since so many powerful 'desktop planetarium' programs are available. While these are very impressive, and often very useful, they don't always provide the information needed by professional astronomers in the most useful form. Furthermore they are generally wedded to a particular architecture and operating system, generally PCs or Apple machines.

I wrote this program to serve my own needs, which are broadly typical of the professional observing community. Many amateurs may find it useful as well. It is written in garden-variety C and has a dead-simple user interface (you type, then it types – no mouse, no graphics, nothing difficult to move from machine to machine). Commands are terse. The code is designed to run on workstation-class machines ubiquitous in professional circles; modifications are necessary to fit it onto PC-class machines (see the notes at the end if this is of interest). There is a new (summer 1994) provision to write calculated results directly to a log file without having to use operating-system output redirection.

The program is designed as follows. You specify information about your site, the coordinates of your object, the date and time, or whatever else is relevant, using very simple commands and a flexible, obvious format. Then you give a command to do calculations and put out results; some of these commands prompt for further needed information. The output commands are explained in more detail later, but the following summary gives some idea of the program's aims.

- `h` The 'hourly airmass' command prints a table of the airmass, hour angle, and other information for each hour during the night. Users tell me that they use this more frequently than any of the other options. It uses the date, the site, and the object coordinates.

- `a` The 'nightly almanac' command tabulates information about a single night, including times of sunset, sunrise, twilight, moonrise or -set, the moon phase, moon coordinates, the moon phase at midnight, sidereal times at midnight and twilight, and other such. It uses only the site information and the date.

- `=` This prints the 'instantaneous circumstances' for your observation; it uses practically all the input information (site, RA and dec, date and time) and tells you the airmass, precessed coordinates, the state of the moon and its modeled sky-brightness contribution, twilight, heliocentric (barycentric, actually) corrections, the parallactic angle, the julian date, and various other stuff. There are little niceties such as a check as to whether a major planet might be near the line of sight, and whether a solar or lunar eclipse is in progress.

- `o` The 'seasonal observability' command is designed to help you accurately assess the 'range of acceptable dates' for observing a given object. It tabulates how many hours an object will be observable at night at less than 3, 2, and 1.5 airmasses from your site; the tabulation is for each new and full moon between two specified dates. Thus this serves as a lunar phase calendar as well.

- `m` This 'major planets' command types out rough positions of the major planets for your site, date, and time, including their hour angles and airmasses.

In addition, there are several special-purpose calculators invoked by two-character commands (the two-character commands are called 'extra goodies'). These list times of events for a variable star, precess batches of coordinates, convert julian to calendar dates, give the offset from ephemeris time, and such.

Specifiying input parameters one-by-one can be tedious, so there are a number of ways to make this more convenient. Sites (which include most of the world's major observatories) are presented on a menu. One can read the system clock with `T` and set the date and time to 'right now' (plus a settable offset, in case you're interested in, say, half an hour from now). This is especially useful at the telescope. There is also provision under extra goodies for reading in a list of objects (in a simple format), presenting this list sorted in various ways, and selecting object coordinates from it. Finally, one can automatically set the coordinates to the zenith for the specified site, date, and time.

There's flexibility as to how you specify dates and times, which comes at a cost in consistency and simplicity. By toggling software switches you can specify times either in UT or local zone time (which can optionally include daylight savings); also, you can apply a date convention by which the evening date applies all night, for nighttime continuity.

Later in this document I give a lengthy and detailed description of the level of accuracy expected for all the calculations. My philosophy has been to compute everything as accurately as I could, consistent with the requirement that the program be self-contained and portable. For example, the precession and sidereal time calculations are very accurate because they are reasonably compact, but the planetary positions are not definitive because that would require the inclusion of rather extensive data tables. Which leads to ...

*** A FEW CAUTIONS: I've made this code as accurate and as generally useful as I could, but before using it for purposes where extreme accuracy is critical, or for locations at extreme geographical positions, it's a good idea to read up on the algorithms and their limitations. And it's always the user's responsibility to be sure the answers are sensible. Be especially careful when the code is recompiled on your local machine; experience shows that compilers can generate different answers from the same code. The examples below can be used to check that your local compiler gives good answers. With these caveats, we proceed to.... ***

1.1 – BASIC USE OF THE PROGRAM.

Before starting, note that you should ideally be able to get going with the program in about 10 minutes *without* referring to this document, using the fast guided tour option and other help text. (I've established this empirically with volunteer astronomer subjects). Nonetheless, going through the program with this document in hand should give a more thorough understanding of the program and draw attention to the many things it can do. This document should be a useful reference, especially for questions of accuracy, generality, and such. I assume that the user is familiar with the concepts of celestial coordinates, sidereal time, and so on; the program has no provision for teaching complete novices!

Computerphobes please note: There's very little you could do which could cause the program to 'run away', and although it can write out log files it doesn't have one open automatically. So you won't hurt anything by mis-typing. The program does not prompt you under most circumstances, which can be disconcerting; but typing a few empty carriage returns in a row will usually point you towards help.

Starting Up

Begin by running the program. How you do this is dependent on your operating system. On a UNIX system, you just type the name of the program, which is likely to be `skycalc`, assuming it is in your current directory or has been installed in your path.

The program first asks you to select an observing site; a little menu comes up which gives single-character codes for a number of major (and minor) observatories. Simply type the single character for the site you want, followed by a carriage return. (Throughout the program, nothing happens until you type a carriage return). The examples computed below are for Kitt Peak (`k`), but you can give whatever you like. Note that the input is 'case sensitive', so `K` is not the same as `k`!. Here's what the menu should look like, though you'll undoubtedly have more menu options:

```
Astronomical calculator program, by John Thorstensen.
```

```
  *SELECT SITE* - Enter single-character code:
      n .. NEW SITE, prompts for all parameters.
      x .. exit without change (current: Kitt Peak)
      k .. Kitt Peak [MDM Obs.]
```

... (and so on ... several different sites are available) ...

```
      l .. Lick Observatory
 Your answer -->
```

If the desired site is not on the menu, type the letter for a new site (`n` in this example) and you'll be prompted for the characteristics of the site. The prompts should be self-explanatory. Note that the longitude and time zone are *positive westward*, unlike the Almanac convention. Also, you must sepcify the longitude in *hours* minutes and seconds, and the latitude in *degrees* minutes and seconds.

The program next attempts to read your computer's internal clock to establish the time and date. Thus the program should wake up with the time and date set to 'right now' * . This may not be what you want, but among all possible times it's perhaps the most likely choice, so it's the default. The machine will tell you what it has set for time and date.

After this, the program sets the coordinates for computations to the zenith for the specified site at the date and time which have just been set. This again may not be what you want, but if you're at the telescope it at least assures that the coordinates are above the horizon, and you can easily change this later. The default coordinate epoch is 1950, so you may notice that the declination of the zenith is in general slightly different from the observatory latitude.

Finally, the program suggests that new or rusty users take the 'fast tour' sequence by typing `f`. I *strongly* suggest that new users go through this 10-minute exercise. The rest of this discussion follows this tutorial introduction.

Specifying a date (`y`*) and getting an almanac (*`a`*)*

The guided tour first suggests you specify an evening date (as year month day) and get the almanac for that date by typing

```
  y 1995 3 22 a
```

(followed, as always, by a carriage return).

The *command-line syntax* of this program, such as it is, is nicely exemplefied here. The `y` command means 'set the date to the following date, expressed y m d'. The reason for the somewhat non-obvious choice of `y` to specify the date is that `d` is used for declination; since the date format starts with the year, this is at least a little bit mnemonic. The `a` command means 'print the almanac information for the presently specified (evening) date'. Note that the commands are case-sensitive,

* This works on all systems I've tested so far, and the c library functions I use for this are supposedly standardized. In case it doesn't work I've built a switch into the source code to disable all the functions which rely on the system clock; if the system clock is disabled, it will tell you at this point and set the time and date to a default value of 2000 Jan 1 at midnight.

so `Y` will not work in place of `y`. The program seldom cares where carriage returns are placed, but doesn't do anything until a carriage return is typed. This command produces the following:

```
Here's the almanac for the currently specified date:
Almanac for Kitt Peak [MDM Obs.]:
long.   7 26 28 (h.m.s) W, lat.  31 57.2 (d.m), elev.  1925 m
Mountain Standard Time (  7 hrs W) in use all year.

For the night of: Wed, 1995 Mar 22 ---> Thu, 1995 Mar 23
Local midnight = 1995 Mar 23,  7 hr UT, or JD 2449799.792
Local Mean Sidereal time at midnight =  11 34 44.2

Sunset (  700 m horizon):   18 43 MST; Sunrise:    6 23 MST
Evening twilight:   20 01 MST;  LMST at evening twilight:    7 35
Morning twilight:    5 05 MST;  LMST at morning twilight:   16 41
12-degr twilight:  19 32 MST -->   5 34 MST; night center:    0 33 MST

Moonrise:    0 53 MST
Moon at civil midnight: illuminated fraction 0.555
0.5 days before last quarter, RA and dec:  17 42 18, -19 53.7

The sun is down for 11.7 hr;  9.1 hr from eve->morn 18 deg twilight.
 4.9 dark hours after end of twilight and before moonrise.

Type command, 'f' for fast tour, or '?' for menu:
```

Note that the date you have given is interpreted as the the *local* date for *evening*; the rest is largely self-explanatory, but a few remarks are in order. The times of moonrise or moonset are reported only if they occur when the sun is down (or close to it). The phase of the moon is printed, together with its illuminated fraction and celestial coordinates; these are computed for local midnight, whether or not the moon is actually up at the time. If the observatory elevation is non-zero, an approximate correction is applied to the times of moonrise and moonset; this is discussed more fully in the section on algorithms and accuracy.

Specifying ra and dec (`r` and `d`); hourly airmass (`h`)

Now let's get a little more specific and consider observing a particular object. The guided tour now suggests you specify an RA and dec and make an hourly airmass table by typing

```
r 15 28 27 d 12 13 14 h
```

Note here that the default epoch for input coordinates is 1950, but you can change the input epoch by typing, say, `e 2000`.

As the example shows, times, right ascensions, and declinations are generally entered as *triplets of numbers* separated by 'whitespace' characters (blanks, tabs, or newlines). Colons do *not* work as delimeters; the spacebar is a lot easier to type, anyway. A little flexibility is allowed in these formats.

The leading parts (hours, degrees, minutes) of right ascensions and declinations can have a fractional part; for instance, RA $19^h15^m00^s$ could be entered as

```
r 19.25
```

and in recent versions you don't have to enter the trailing zeros, provided your next character is a valid command (such as `=`), and there is at least one blank following your number. To enter a negative declination, just make the first number negative, as in

```
d -0 18 30
```

(-0 is correctly handled to give a negative declination, but there cannot be any space between the minus sign and the number following.)

The `h` command, which generates the hourly airmass table, first prompts you for the `Name of object`. The reason for this is that you may wish to redirect output from the program (discussed later) to make a hard copy; the name then serves to label the output. The name serves only as a label, so you can give a single character if you want. Here's what the output looks like:

```
*** Hourly airmass for Flapdoodle's Variable Nebula ***

Epoch 1950.00: RA  15 28 27.0, dec  12 13 14
Epoch 1995.22: RA  15 30 35.7, dec  12 04 01

At midnight: UT date 1995 Mar 23, Moon 0.56 illum,  45 degr from obj

  Local      UT      LMST      HA     secz   par.angl. SunAlt MoonAlt

  19 00     2 00     6 34    -8 57   (down)   -43.5    -5.2     ...
  20 00     3 00     7 34    -7 57   (down)   -50.8   -17.8     ...
  21 00     4 00     8 34    -6 56   (down)   -55.7     ...     ...
  22 00     5 00     9 34    -5 56    8.033   -58.8     ...     ...
  23 00     6 00    10 35    -4 56    2.947   -60.1     ...     ...
   0 00     7 00    11 35    -3 56    1.857   -59.6     ...     ...
   1 00     8 00    12 35    -2 56    1.412   -56.5     ...    -0.3
   2 00     9 00    13 35    -1 56    1.194   -48.5     ...    10.5
   3 00    10 00    14 35    -0 55    1.091   -30.4     ...    20.4
   4 00    11 00    15 35     0 05    1.064     3.0     ...    28.8
   5 00    12 00    16 36     1 05    1.102    34.4     ...    34.8
   6 00    13 00    17 36     2 05    1.220    50.3    -6.5    37.8
```

Each line shows the local time, the UT, the local mean sidereal time, and the object's hour angle; the next quantity $\sec z$, the secant of the zenith angle, is essentially the same thing as the airmass. The notation `(down)` in this column means the object is below the horizon; `v.low` will occasionally appear in this column, meaning that the object is so near the horizon that $\sec z$ will overflow the space provided for it. Note that the last two columns give the altitude of the sun and moon; the sun is printed if it is higher than $-18°$, and the moon if it is higher then $-2°$. Otherwise ellispes (...)

are printed in those spaces. Also notice that the line giving the UT date at (local civil) midnight also gives the moon's illuminated fraction and its angular distance from the object. These are computed for local midnight, whether or not the moon is actually up then.

A Word about Hard Copy and Log Files

Users tell me that the `h` command is the most commonly used output feature, and that it's especially nice to have hard copies of the tables for your main targets. This therefore seems like a good place to mention two ways to direct the program's output to a file: one can either redirect output using general operating-system features, or use a newly installed *log file* feature. The log file should be the more convenient of the two possibilities.

The log file feature (if enabled – see next paragraph) is invoked from the 'extra goodies' menu by typing `xL`. The big `L` in this case is meant as a mnemonic that something big is happening (generally, I reserve upper-case for commands which change more than one thing at a time). You're prompted for the name of a log file. *When the log file is open, almost all the program's output (except for prompts) is also written to the log file.* If you print out the online documentation – such as the menu and fast tour – this goes to the log file as well. The log file has minor format differences from the terminal output to make it easier to read. If you type `xL` again while the log file is open, it closes the log file, so `xL` acts as a toggle. The log file is opened with 'append' permission, so if it exists already new output is written to the end.

Because there may be circumstances – public accounts or some such – in which one does not want to grant users permission to write files, you can disable the log file option by turning off a preprocessor switch in the source code and recompiling. The switch is called `LOG_FILES_OK`.

More general output-redirection using the operating system is not as simple, and details are system-dependent. You'll generally have to prepare a list of commands to feed into the program. Here's a sample of what the command input file might look like in a typical case for the `h` command; the comments are for human legibility and are not to be included.

```
(input)                 (comment)

k                       (selects Kitt Peak, assuming it's in site menu).
y 1990 10 20              (date)
e 2000                  (select epoch of input coordinates)
r 19 19 19              (coordinates .. ra and dec).
d 2 2 2
h                       (hourly circumstances command)
Wholeflaffer 9         (name of object .. the output will follow.)
r 20 20 20              (for next object)
d 12 12 12
h
Flapdoodle's Variable Nebula
```

and so on... `r, d, h,` followed by the name, until..)

```
Q              (exit program).
```

(Note: A previous version of the `h` command prompted for the number of hours to print; this is now computed automatically, so old scripts designed for the previous version will have to be revised slightly.)

Instantaneous Circumstances; the `=` *Command*

The next action suggested in the guided tour is to specify a time of day, and then display the instantaneous circumstances by typing, for instance,

```
t 4 50 0 =
```

By default, the time you enter is taken to be the local time, but this can be changed to UT with the `g` option (below). Also by default, a morning time (such as this one) is interpreted as referring to the morning of the date *after* the specified date; this way, morning and evening times refer to the *same night*. This can also be changed, using the `n` ('night-date') option. These are all explained later.

The `=` causes the instantaneous circumstances to be displayed; for the present parameters, the output is:

```
W Long (hms):  7 26 28.0, lat (dms): 31 57 12, std time zone   7 hr W

Local Date and time: Thu, 1995 Mar 23, time   4 50 00.0  MST
   UT Date and time: Thu, 1995 Mar 23, time  11 50 00.0
Julian date: 2449799.993056   LMST:  16 25 31.8

Std epoch--> RA: 15 28 27.0, dec: 12 13 14, ep 1950.00
Current  --> RA: 15 30 35.7, dec: 12 04 01, ep 1995.22
HA:   0 54 56; sec.z =    1.091
altitude  66.44, azimuth 215.51, parallactic angle 30.3   [-149.7]

The sun is down; there is no twilight.
Moon : 17 52 07,-20 04.7, alt  34.0, az 155.3; 0.537 illum.
0.3 days before last quarter.  Object is  47.4 degr. from moon.
Lunar part of sky bright. =  20.7 V mag/sq.arcsec (estimated).
Barycentric corrections: add  305.9 sec, 18.30 km/sec to observed values.
Barycentric Julian date = 2449799.996597

Type command, 'f' for fast tour, '?' for a menu:
```

Notice all that has been computed!

After a brief summary of the site information, the next block establishes the time in various systems. The date and time are given in both local and UT. If daylight savings time is selected, a recipe which should be appropriate to the site is used to select whether the local time is reported in DST or Standard. Note that the label on the local time will have either an 'S' or a 'D' as the second character, to indicate standard or daylight. LMST stands for the *local mean sidereal time*, which is essentially the local sidereal time. It disagrees with the true hour angle of the equinox by the 'equation of the equinoxes', caused by nutation; this amounts to a couple of seconds at most.

The next block refers to the object. The object's coordinates are reported both for the 'standard epoch', which is set with the `e` option, and for the mean equinox of date. The program 'wakes up' assuming that input coordinates are for equinox 1950. Proper motions may be included (see below).

The quantity $\sec z$ is (for most purposes) nearly the same as the airmass; the difference grows large only near the horizon. If the object is at a particularly large airmass, or below the horizon, a comment is printed. If the object's airmass is very large, it is not printed to avoid overflowing the space provided. The *altitude* is 90 degrees minus the zenith distance, uncorrected for refraction; the *azimuth* is (as usual) measured from the north through the east. The *parallactic angle* is the position angle (measured N through E at the object) of the arc that connects the object to the zenith, or loosely speaking, the position angle of 'straight up'. This is useful for setting a spectrograph slit to catch the dispersed light. (See Filippenko, 1982, PASP, 94, 715 for a discussion). The parallactic angle may change sign at the meridian, but actually varies smoothly. Because some applications (e.g., the placement of a spectrograph slit) are indifferent to a 180-degree shift in the parallactic angle, the angle ± 180 degrees is reported in square brackets.

If the moon could be interfering (higher than 2 degrees below the uncorrected geometrical horizon), its phase, altitude (not corrected for refraction), fraction illuminated, approximate RA, dec, altitude, and azimuth are reported. Also, the angle subtended at the observer by the object and the moon is reported. If the moon is more than two degrees below the horizon, it is reported to be 'down' and only its phase is printed. If both the moon and the object are above the horizon, *and* the sun is more than 9 degrees below the horizon, an estimated value of the moon's contribution to the night-sky brightness is given; this is obviously only approximate, and only holds under ideal conditions. For comparison, a dark site has about 21.5 V magnitude per square arcsec, but this varies considerably with solar activity and such.

If the sun is at a geometric altitude $> -18^\circ$ but below the horizon, twilight is reported. The zenith distance at which the sun's upper limb reaches the horizon is taken to be 90.83° if the observatory elevation is zero; the extra 0.83° account approximately for the refraction and average angular semidiameter of the sun. A further correction appropriate to a sea-level horizon is added if the site's elevation is nonzero. In twilight, an estimate of the brightness of twilight at the zenith is reported; these numbers appear to match rather well the behavior of twilight in blue light. In visual or red, the enhancement due to twilight may be rather fainter. If the sun's upper limb is above the horizon, it is reported to be 'up'. In twilight or daylight the RA, dec, altitude, and azimuth of the sun are given. This altitude is not corrected for refraction. If the sun's zenith distance is greater than 108° (or more than roughly eighteen degrees below the horizon), it is reported to be 'down'.

Another feature checks to see if the position you've specified is within 3° of the computed (low-precision) position of any major planet. If it is, the program warns you. This way you won't try to set on your 98th magnitude object only to find that Jupiter is 5 arcminutes away! Yet another feature reports if an eclipse of the sun or moon in progress. The accuracy expected of these predictions is discussed later.

The `o` *command - Observability through a season*

The next suggestion in the guided tour is to explore the observability of your object through an observing season, by typing `o`. The output is designed for use *before* you apply for telescope time – it supplies you with accurate information to allow you to decide the 'range of acceptable dates' by printing a summary of the observability of your object (as specified by the RA, dec, and epoch)

You are first prompted for the range of dates to cover, (in standard y m d form). A 6-month span fits in a standard 24-line screen, if that it important. It next asks for the altitude of the sun to be used for twilight; $-18°$ is standard astronomical twilight, but this is very dark, so you may wish to relax this condition some if you can live with a little sky light. Finally, it prompts for an object name (as in `h` above), simply to use as a label for any hard copy you might make. The object we've been working with (at about 15 hours) would be expected to culminate at midnight in May; specifying `1995 2 1` and `1995 9 1`, and standard twilight gives the following output:

```
           *** Seasonal Observability of Veeblefetzer's Star ***

        RA & dec:  15 28 27.0,  12 13 14, epoch 1950.0
   Site long&lat:   7 26 28.0 (h.m.s) West,  31 57 12 North.

   Shown: local eve. date, moon phase, hr ang and sec.z at (1) eve. twilight,
   (2) natural center of night, and (3) morning twilight; then comes number of
   nighttime hours during which object is at sec.z less than 3, 2, and 1.5.
   Night (and twilight) is defined by sun altitude < -18.0 degrees.

    Date (eve) moon      eve           cent          morn      night hrs@sec.z:
                       HA  sec.z      HA  sec.z      HA  sec.z     <3   <2  <1.5
   1995 Feb 14   F -10 44  down     -5 37   5.2    -0 30   1.1    4.5  3.6   2.7
   1995 Feb 28   N  -9 39  down     -4 44   2.6     0 11   1.1    5.1  4.3   3.4
   1995 Mar 16   F  -8 24  down     -3 45   1.7     0 55   1.1    5.9  5.0   4.1
   1995 Mar 30   N  -7 18  down     -2 54   1.4     1 31   1.1    6.5  5.6   4.7
   1995 Apr 14   F  -6 06  11.2     -1 59   1.2     2 08   1.2    7.1  6.3   5.3
   1995 Apr 28   N  -4 57   3.0     -1 06   1.1     2 45   1.4    7.7  6.9   5.9
   1995 May 14   F  -3 38   1.7     -0 04   1.1     3 30   1.6    7.1  7.1   6.4
   1995 May 28   N  -2 30   1.3      0 52   1.1     4 14   2.1    6.7  6.6   5.7
   1995 Jun 12   F  -1 21   1.1      1 54   1.2     5 08   3.4    6.3  5.5   4.5
   1995 Jun 27   N  -0 18   1.1      2 56   1.4     6 10  13.3    5.2  4.4   3.5
   1995 Jul 11   F   0 34   1.1      3 54   1.8     7 14  down    4.4  3.6   2.6
   1995 Jul 26   N   1 22   1.1      4 54   2.9     8 26  down    3.6  2.8   1.8
   1995 Aug 9    F   2 02   1.2      5 48   6.5     9 34  down    2.9  2.1   1.2
   1995 Aug 25   N   2 44   1.4      6 48  down    10 52  down    2.2  1.4   0.5
   1995 Sep 8    F   3 19   1.5      7 39  down    11 59  down    1.6  0.8   0.0
   Listing done.  'f' gives tutorial, '?' prints a menu.
```

Because observing time requests are so intimately tied to lunar phase, the dates selected are those of full and new moon; they are selected to be those (local evening) dates on which new or full moon fall within 12 hours of the center of the night. The tabulation starts with the lunation before your specified starting date. At each date, the object's hour angle and airmass (actually $\sec z$) is given (1) at evening twilight, (2) at the natural center of the night, and (3) at morning twilight. The 'natural center' is time of the sun's lower culmination (when its hour angle is 12 hours); in general it differs from local clock midnight because of location in the time zone, daylight savings time, and the equation of time. Finally, the last three columns give the number of hours during the night (that is, past twilight) for which the object is at airmasses less than 3, less than 2, and less than 1.5. These limits are arbitrary, but representative of poor, marginal, and good observability. Circumpolar objects can be observable both at the beginning and the end of a long winter night; the code appears to tally the observable hours properly.

In high latitudes, twilight does not occur in midsummer; in this case, `twi.all.night!` appears in the columns for position at evening and morning twilight. At extremely high latitudes, the sun can remain below the specified twilight altitude all day, and these columns then contain information for the times at which the sun is ± 12 hours from its lower culmination.

Looking at current parameters with `l`

At this point we've set a fair number of parameters. While many of the the current parameters are printed in the output from `=`, others are implicit, and the display is crowded, so they're hard to keep track of. Thus the `l` ('look') command simply prints out a nicely-formatted list of input parameters. Its output is

```
Current INPUT parameter values:

      DATE: 1995 Mar 22
      TIME:  4 50 00.0
NIGHT_DATE:  ON    -- date applies all evening & next morning.
  UT_INPUT:  OFF   -- input times taken to be local.
   USE_DST:   0    -- Standard time in use all year.

            RA:  15 28 27.00
           DEC:  12 13 14.0
   INPUT EPOCH:    1950.00
PROPER MOTIONS:  OFF

SITE: Kitt Peak [MDM Obs.]
      E.longit. = -111 37.0, latit. =  31 57.2 (degrees)
      Standard zone =   7 hrs  West
      Elevation above horizon =  700 m, True elevation = 1925 m
```

This is particularly useful for keeping track of the `g` and `n` commands, which cause the interpretations of time and date to *toggle* between different cases. Because the effect of each of these commands depends on the status when they are executed, it's helpful to be able to look at their state without doing anything else. Also, the site latitude and longitude are converted here to a format which exactly matches the numbers in the *Astronomical Almanac* observatory list, to make it easy to check them.

1.2 – OTHER CALCULATOR-MODE COMMANDS.

Quitting the program – `Q`

This stops the program gracefully. The `Q` must be upper-case – this should avoid accidents well enough.

Printing a menu – `?`

This causes the following menu to print out.

```
Circumstance calculator, type '=' for output.
Commands are SINGLE (lower-case!) CHARACTERS as follows:
 ? .. prints this menu; other information options are:
i,f . 'i' prints brief Instructions and examples, 'f' fast tour
 w .. prints info on internal Workings, accuracy & LEGALITIES
TO SET PARAMETERS & OPTIONS, use these (follow the formats!):
 r .. enter object RA, in hr min sec,  example: r 3 12 12.43
 d .. enter object Dec in deg min sec, example: d -0 18 0
 y .. enter date, starting with Year   example: y 1994 10 12
t,T: t = enter time, e.g.: t 22 18 02 [see 'g' and 'n']; T = right now+
 n .. *toggles* whether date is used as 'evening' (default) or literal
 g .. *toggles* whether time is used as Greenwich or local
 e .. enter Epoch used to interpret input coords (default = 1950)
 p .. enter object Proper motions (complicated, follow prompts).
 s .. change Site (again, follow prompts).
 l .. Look at current parameter values (no computation).
TO CALCULATE AND SEE RESULTS, use these commands:
 = .. type out circumstances for specified instant of time, ra,dec
 a .. type out night's Almanac for specified (evening) date
 h .. type out Hourly airmass table for specified date, ra, dec
 o .. tabulate Observability at 2-week intervals (at full&new moon)
 m .. Major planets -- print 0.1 deg positions for specified instant
 x .. eXtra goodies! (galact./eclipt., var star predicts, precess.)
 Q .. QUIT .. STOPS PROGRAM. --->
```

I've found in testing that this menu is good for reminding advanced users of commands, but poor for teaching new users how to run the program. Hence it is not mentioned in the introductory banners, but rather the user is referred to the fast guided tour, invoked with `f`.

A word about 'extra goodies' – `x`

You'll notice an `extra goodies` option on the main menu. This option hides some commands which are unlikely to see much use, and also some more complex commands which I thought it wisest to hide from novices. Putting them here keeps the main menu short enough to fit into a 24-line display. If you type `x?` you'll get a summary which describes these selections. The extra goodies command level does not loop, but drops you immediately back into the main menu whatever you do, so you will have to prefix any new extra goodies command with another `x`.

Setting the time to now – `T`

An UPPER CASE T sets *both* the time and the date using your machine's system clock. (As noted earlier, this option may be switched off at compile time if there is some problem with this.) There's an interesting twist here: you are prompted `Set how many minutes into the future?  :`. Answering `0` sets the time and date to right now; any other number sets the time and date into the future (or past for negative numbers). This lets you quickly answer questions such as 'Can I get to this object half an hour from now when I've finished with exposure I'm working on?'.

The internal actions of this option are modified by the `g` and `n` options (see below). The program should do the right thing and set the date and time to reflect the present (with whatever offset you specify) as expressed *in the prevailing time and date convention.*

Note that this option does *not* cause the time to be continually updated; the value of the time set by `T` remains in effect until you set the time to some other value.

Changing the Site with the command `s`

You can change sites by typing the letter `s` and answering the prompts. When you do this, you will be given a menu of single-character site codes from which to choose, just as when you started the program. Your local version of the program should be customized to offer the most common choices for your institution. To choose a site, just type the letter (be sure to use the correct lower or upper case) and hit carriage return. You can also specify a site not on the menu by typing `n` (or the appropriate character in your customized version). If you select one of the 'canned' sites, all the parameters (latitude, longitude, time zone info, etc.) will be changed to their standard values for that site.

If you want a site which is not on the menu, you'll have to give all its parameters. Otherwise one would risk of changing the parameters piecemeal and having some parameters which are appropriate to the site and others which are not. You'll need to know the latitude of your site in degrees, minutes, and seconds, and the *west* longitude in *hours*, minutes, and seconds. (Like Jean Meeus, I dislike the 'east longitude' convention). You'll also need the time zone in hours west of Greenwich (*e.g.*, Pacific is 8). You can specify Eastern hemisphere sites by giving negative numbers for the longitude and time zone. You'll also be prompted for the site's elevation above sea level, which affects certain quantities very slightly, and its elevation above its horizon, which is used only to adjust rising and setting times.

The last parameter prompted for is whether *daylight savings time* is to be used in converting between local and UT. There are several options given here. Typing `0` ignores daylight time. Typing

`1` invokes the conventions in use in the United States; daylight savings starts on the first Sunday in April and ends on the last Sunday in October from 1986 on, and from the last Sunday in April before that. (This ignores various wrinkles during wars, energy crises, etc.). Typing 2 gets you the Spanish (Continental?) convention, with daylight savings from the last Sunday in March to the last Sunday in September. Negative numbers are used for southern sites; typing `-1` gets you the Chilean convention (off daylight savings the second Sunday in March, back on the 2nd Sunday in October), and `-2` gets the Australian convention. Implementing other prescriptions would require straightforward modifications to the source code. The presently available prescriptions all assume that the time changes at 2 AM as reckoned in the time preceding the change, as is standard in the US.

Naturally, you should be sure that you have the correct parameters for your site. The `l` command lays out the site parameters neatly for your inspection, and many are echoed with other output.

UT time input and 'night dates' – `g` *and* `n`

Typing the letters `g` or `n` switches between various options for the interpretation of input times and dates.

The `g` command switches between input in UT and in local time. The program wakes up assuming that dates and times are input in local time; typing `g` makes the program assume that the input date and time are in UT; a little message is printed telling you this. Typing `g` again switches back to local, and so on. Whichever time is assumed for the input is printed first on output. Notice that, when you type `g`, the *current time changes*; the input time and date have their same numerical *values*, but are now interpreted differently! A message is printed to remind you of this.

Similarly, the program awakens assuming that the date you specify is to be interpreted as the *evening* date for the entire night (this is the 'night date' condition). For example, if you print out an almanac for the *night* of October 20, and then specify a time after midnight (2 30 00, say) and type `=`, the circumstances printed are those applying on the *morning* of October 21. The reason for doing this is to maintain some parallelism with the almanac, which prints the phenomena for a given night. Typing `n` once switches this option off, so the current date is interpreted literally; typing `n` again switches it back on (unless you are in UT mode), and so on. This may seem confusing at first, but it should be less confusing than the alternative. It is at least always possible to interpret the output unambiguously; the times and dates printed there are generated internally directly from the JD, so they should always be reliable.

The `g` and `n` commands interact. Going to UT input automatically turns off the 'night date' option, since UT dates should always be interpreted literally. You are also prevented from turning on the night date condition when UT input is in effect.

Coordinate epoch – the `e` *command, and* `xb` *batch precesssion*

This is pretty simple; typing `e` followed by an epoch sets the epoch for your input coordinates. Setting this option does not in itself cause any actual coordinates to be transformed, but rather affects the *interpretation* of the input coordinates when computations are done. The `=` and `h` commands (and others) do these computations, and show both the input coordinates and the coordinates in the epoch of date.

There is a command hidden in the `eXtra goodies` menu, `xb`, which transforms a batch of coordinates from one epoch to another. This prompts for input and output epochs, and then for coordinates in the usual format. It keeps going until you give it a negative RA (such as `-1 0 0`). The batch precession is isolated from the rest of the program – it does not affect any parameter values.

The accuracy of the precession calculations is discussed later; note that a completely kosher transformation from B1950 to J2000 includes adjustments to the proper motions (since the reference frame was refined), and the current program does not do this.

Proper motions ... the `p` *command.*

If you type `p` you will be prompted for annual proper motions of the object; answer the prompts. The specification of proper motion is complicated because there are (at least!) two conventions in use for the units of the proper motion in RA. One is the annual change of the RA itself, generally given in seconds of time per year; this is used in the SAO Catalog. The other is the east-west motion in seconds of arc on the sky, which is the first times $15 \cos \delta$. The program will accept input of either type; if you give seconds of time per annum, you must follow your value with an `s`, and if you give seconds of arc you must give an `a`. The value is converted and passed internally in the first (time) convention. Declination proper motions must always be entered as arcsec per annum.

If either of the proper motions are nonzero, the output of `=` will display

- the original coordinates in the standard epoch and equinox
- the coordinates updated for proper motion only (current *epoch*, but standard *equinox!*)
- the coordinates updated for proper motion *and* precession (current epoch *and* current equinox).

as well as the proper motions used. The reason for doing this is that with most modern telescopes the coordinate readout can be set to a standard equinox, but the actual sky is (of course!) always in the present epoch, regardless of what coordinates you apply to it. So it's useful at times to displaying the updated position without changing the equinox. Note that the proper motions are not computed with perfect rigor; the current RA is just the old RA plus $\mu \Delta t$, and similarly for the dec. This is inaccurate very close to the pole or over very long intervals of time.

The `xc` *command – coordinate conversions.*

Typing `xc` causes the galactic and ecliptic coordinates to be printed. (As of this writing, there is also a `c` at the main program level which still does this, but it's not advertised on the main menu, in order to keep the main menu to 24 lines). The galactic coordinate algorithm complies strictly with the IAU definition, which is specified in 1950 coordinates. If the input coordinates are in a different epoch, they are precessed internally to 1950 before being converted to galactic. Both conversions work correctly over the entire sky. The inverse conversions are not implemented.

The `xj` *command – calculate calendar dates from Julian dates.*

The main program converts calendar to julian dates internally, and prints out julian dates with, among others, the `=` command. It's sometimes useful to have the inverse, which converts julian to calendar, and the 'extra goodies' command `xj` does this calculation. The command loops until a negative julian date is given. The routine expects all the leading digits of the julian date. If the input is a true julian date, the output is a UT date. The date in the main program is unaffected by this command (see `xJ` below).

The `xJ` *command – Set to a Julian date.*

Again, this converts Julian to calendar dates, but *it also resets the date and time in the main program* to the appropriate values. An upper-case `J` is used because two quantities are reset. The routine takes into account the current input conventions (toggled by `g` and `n`), so the Julian date computed by immediately typing `=` should reproduce the Julian date you have specified. [There is one almost unavoidable **bug**; if daylight savings time is used, then during the double-valued hour when daylight savings time switches back to standard time, the program interprets the input time by default as standard time; a JD during the final hour of daylight time will create a time which will be interpreted later as standard. A prominent warning is printed in this rare case.] Input outside the calendrical limits (1901 – 2099) is rejected. Unlike the calculator-like command `xj`, this one doesn't loop.

`xd` – *Show the value of* TDT − UT.

It's occasionally useful to know the difference between UT (based on the earth's rotation) and TDT (a uniform timescale – see the Algorithms and Accuracy section for a more complete explanation). This computes an approximate value of this quantity for the currently specified date. The approximations used should be good to better than a second from 1900 to 1994, and get increasingly more uncertain in the future (because of the unpredictability of the earth's rotation).

The `m` *command - print a table of the major planets*

As one might expect, this prints a table of the RA, dec, hour angle, secant z, altitude, and azimuth of each of the major planets, as well as the sun and moon. The planetary positions are only modestly precise; their pedigree and accuracy are explained later. The sun and moon calculations are useful for finding their positions for times when they are below the horizon (or past twilight for the sun), and would therefore not be printed with the `=` command. The output is as follows:

```
W Long (hms):  7 26 28.0, lat (dms): 31 57 12, std time zone   7 hr W

Local Date and time: Thu, 1995 Mar 23, time   4 50 00.0  MST
   UT Date and time: Thu, 1995 Mar 23, time  11 50 00.0
Julian date: 2449799.993056   LMST:  16 25 31.8

Planetary positions (epoch of date), accuracy about 0.1 deg:

             RA       dec       HA       sec.z     alt   az

Sun    :   0 08.7     0 57    -7 43      -2.77   -21.1   74.8
Moon   :  17 52.1   -20 05    -1 27       1.79    34.0  155.3

Mercury:  23 01.3    -8 46    -6 36      -4.74   -12.2   92.8
Venus  :  21 47.2   -13 45    -5 22      86.46     0.7  106.7
Mars   :   9 06.7    20 11     7 19     -11.61    -4.9  297.5
Jupiter:  16 56.1   -21 49    -0 31       1.71    35.7  171.2
Saturn :  23 15.6    -6 43    -6 50      -4.09   -14.1   89.1
Uranus :  20 08.6   -20 41    -3 43       3.85    15.0  126.8
Neptune:  19 48.3   -20 33    -3 23       3.15    18.5  130.2
Pluto  :  16 04.1    -6 41     0 21       1.29    51.0  188.5 <-(least accurate)
```

It's entertaining to note that if your location is on the site menu, you can get a table of planets for 'right now' by starting the program, giving your site's letter, and typing `m` – counting carriage returns, this is four keystrokes.

Setting to the zenith with `xZ`

This command simply sets the right ascension and the declination to those of the zenith for the currently defined date, time, and site. It is a capital letter because it changes two quantities at once. The coordinates of the zenith are precessed from the present epoch (for which they are just the sidereal time and the latitude) to the standard input epoch defined by the `e` command.

Object lists – the `xR, xl, xN, and xS` *commands.*

Overview. These commands were added in 1994 February; they allow one to read coordinates from a file, list them, and select from them using various criteria. These capabilities are quite powerful but add some complexity, so the commands are hidden among the 'extra goodies' to avoid intimidating beginners.

At their simplest, these commands allow you to set the RA and dec to those of a given object, without having to type the RA and dec into the program. But they are far more powerful than this; for one thing, the commands to display and select from the object lists also give the *hour angle and airmass* for each object at the currently defined site, date and time. This was designed for use at the telescope – after setting the date and time to the present (with `T`, perhaps) you can quickly scan a list of objects and select one which is well-placed for observation. The `xS` command goes further and presents the list sorted according to various criteria.

The *file format* for objects is straightforward. Files are expected to contain one object per line; the reading is done one line at a time, so an error in one line does not propogate to the next. Here's an example of a correctly formatted line:

```
v1727_cyg 21 29 36.2 47 04 08 1950.  18.1 binary xrs - 5hr
```

The data fields in each line must be separated by blanks or tabs, but otherwise are free format. The first field is a *name*, which must be less than 20 characters and cannot contain any blanks. It is helpful if you keep the names simple so you can remember them exactly. Next comes the RA in hours, minutes and seconds (three fields), then the dec in degrees, minutes, and seconds (three more fields). All these numbers are read as floating point, so for example `19 30.5 0` would be equivalent to `19 30 30`. If the declination is negative the first character of the degree field must be a minus sign, and there must be no space between the minus sign and the remainder of the number. A `-0` declination is handled correctly. In the eighth mandatory field is the coordinate epoch. Finally, the ninth field many contain an *optional user-supplied floating point number*, which might be a magnitude or perhaps a priority for observation. Any further entries on the same line are ignored, so you're free to put notes or other information there (as above). This information will not be read at all; it's only there for your own benefit (say, for a printed copy of the list).

If the program does not successfully read the eight mandatory fields from each line, a complaint is printed and the line is ignored. If the optional user-supplied floating point number is not supplied, it is automatically assigned a value of 99.9. This choice is arbitrary, but it's a rational choice if you're using the field for magnitudes, and it doesn't crowd any of the later displays.

You are allowed up to 499 objects in your list. If you want more, you can change the defined constant `MAX_OBJECTS` in the source code and recompile. The information stored for each object amounts to something like 48 bytes (depending on your machine), so the 499-object limit is about 24 k; most users could expand this without running into memory limitations.

The `xR` command reads objects from a file; it briefly reviews the file format and then prompts for the name of the input file. If you specify the filename `QUIT` (all upper case), it does not attempt to open a file. If you specify a file which the program cannot open, it complains and exits back to the main command level. If the file does open, but you already have objects in memory, it asks you whether you would like to append the new objects or replace the old ones with the new ones. The program then reads the file, complaining if it finds any obvious anomalies (blank lines, non-number in fields supposed to be numbers, or whatever). Finally, it reports how many objects you have, closes the input file, and returns. If you've filled up to the maximum number of objects, it warns you.

Now the fun begins!

The simplest thing you can do with the object list is type out some of the contents; that's done with the `x l` (list) command. You're prompted for the first and last items to print out (by number in the list). The program then prints out the presently defined date and time – for which the hour angles and airmasses are computed – and then simply types out the information for each object. The last two columns give the hour angle and airmass ($\sec z$) of each object.

The commands `xN` and `xS` are upper-case letters because they cause more than one quantity to change at once. `xN` searches for an object by name (it must be an exact match, including the upper or lower case of any letters), and if a match is found the program *sets both the RA and dec to that object.* If the epoch of the object's coordinates in the list is different from the currently defined input epoch (the quantity controlled by the `e` command), the object's coordinates are precessed to the input epoch and the coordinates are set to the precessed values (that is, the program handles this correctly. It would have been possible instead to reset the input epoch as well as the RA and dec, but this would have been even more confusing.)

Like the `xN` command, the `xS` command resets the RA and dec (with precession if need be, as above), but now the user gets to select interactively which object to choose. Ten objects at a time are presented, and the user selects an object (and sets the RA and dec) by giving its number on the list. Typing `m` gives the next 10 objects, while `q` quits the search without assigning coordinates. The search continues until one selects and item or quits, or until the list is exhausted.

The power – and fun! – of this command comes in the order in which the objects are presented – the objects are `sorted` according to various criteria (hence the choice of letter). There are (at present) five different options for the sort:

`xS1` sorts the objects in order of increasing distance from the presently defined RA and dec – it's a 'find nearest'. There are many ways to use this. If you set to the zenith with `xZ`, the objects will be presented in order of zenith distance. If you have an object on your list whose coordinates you remember roughly, but you don't remember exactly what you called it, you can find it quickly by setting to the rough coordinates and then finding the exact match. You can also use it to find an object close to the one you're observing to avoid spending too much time slewing or to match airmasses (but see option `xS3` below).

`xS2` sorts the objects in order of the absolute value of the hour angle. Therefore it shows you which objects are closest to the meridian at the presently defined moment of time.

`xS3` sorts the objects in order of proximity in airmass to the present coordinates. This could be useful to photometrists and infrared astronomers who may wish to match the airmass between program and standard star observation as closely as possible.

`xS4` is especially for people chasing objects into the west – it queries for a maximum acceptable airmass, then sorts objects in order of how many minutes it will be before they reach this airmass. Thus you can see exactly how urgent it is to get to each object.

`xS5` sorts the list in order of the optional user-defined number. If this is a magnitude, it will be in order of increasing magnitude; it may be especially useful to put a priority in this field.

Naturally, only limited information can be given about each item. After you've selected coordinates, it's advisable to type `=` to get a complete listing of the observability information. This is especially true if the moon is up – the object you've selected could be right next to the moon, or even occulted!

1.3 – ALGORITHMS, ACCURACY, AND LIMITATIONS.

Calendar and times.

The time arguments for most of the routines are Julian dates, implemented as double-precision floating point numbers. If your machine's double-precision mantissa isn't reasonably long, you can run into serious inaccuracy. Digital's VAX machines express a JD to a few milliseconds accuracy, but this should be checked when the code is ported to another architecture. Calendar dates and days of the week are derived from a truncation of the Julian date, which is the same in each case, so they should always agree.

As noted earlier, the program makes various transformations to account for zone time, daylight savings time, and such. A subtler issue is the actual timebase which is used for the input time. The distinctions between UT, UTC, TAI, TDT are made authoritatively in the *Astronomical Almanac*, but are widely ignored by astronomers, so I'll explain them briefly here; this isn't an authoritative discussion, but I hope it's essentially correct. *Universal Time*, or UT is Greenwich time based on the true phase of rotation of the earth. The earth's rotation gradually slows with time, and it is sufficiently unpredictable that UT can't be determined accurately until after the fact. There are a few minor variants of UT based on the state of the data reduction in this determination. TAI is *International Atomic Time*, which is the best realizable uniform timescale. UTC, the famous *coordinated universal time* broadcast on WWV, is a compromise between these; it follows UT approximately, but is maintained an integer number of seconds away from TAI by the insertion of an occasional 'leap second'. UTC and UT should be maintained so that they always agree to within 0.9 sec. Finally, TDT is *Terrestrial Dynamical Time*; this is another uniform timescale offset for historical reasons I don't understand by a constant 32.184 seconds from TAI. Strictly speaking, before 1983 the apparent conceptual equivalent of TDT was called Ephemeris Time (ET); I'm unclear as to the difference between ET and TDT. On long timescales UT drifts parabolically away from TDT (or its rough equivalent, ET); a perusal of pp. K8 and K9 of the 1995 *Almanac* shows that they were equal in 1870 and 1902, and that the difference $\Delta T = TDT - UT$ has now reached about a minute.

Because calculations of solar system objects should be based on a uniform time scale, the 'argument' of these calculations is generally TDT. But I ignore ΔT in the planetary calculations, because the planets move rather slowly, and the planetary theory used here is relatively primitive. However, TDT is used for the moon calculation (where it is just significant because the moon moves so quickly) and the sun (where at present it changes the answer by about 3 arcsec). From 1900 to 1993 the values used are based on linear interpolations on 5-year intervals in the *Almanac*. Accuracy appears to be less than a second when compared to the annual values tabulated in the *Almanac*. After 1993, the correction used is only a guess, which is linearly extrapolated from present-day values. A parabolic extrapolation might be better, but the behavior in the past has often been rather erratic so this seems adequate.

The calendrical routines break down before 1901 and after 2100. Input outside those dates causes the program to become uncooperative until you set a date inside the allowed range. While it would be a simple matter to extend the calendrical routines, I worry about the wisdom of this because I have not tested the accuracy of the celestial calculations far outside of the present. The routine which converts julian date back to calendar date, which can be accessed directly with the `xj` command, has a wider range of validity; it agrees with the *Astronomical Almanac* (1995; page K4) from 1600 to 2100 at least.

When printing the phenomena for a given night, the program assumes implicitly that zone time at least grossly approximates local time. Thus working from a California location (zone = 8, or Pacific time) and attempting to get times printed as UT by giving a standard time zone as 0 will give peculiar behavior. The `g` option allows input in UT.

As previously noted, daylight savings time is implemented using hard-coded algorithms to determine the dates on which the clock time changes. If your location uses some different algorithm, you'll have to put it into the source code. Note that if you use daylight time and (as is the default) specify your input times in local time, difficulties arise when daylight and standard times switch. When daylight savings switches back to standard time ('fall back'), the numerical value of the time repeats for an hour; going the other way ('spring forward'), there is an hour of non-existent local times. The program handles these conditions as follows. If the specified time is within about 12 hours of the switch, a warning is printed. If you specify a time during the hour which is skipped when daylight savings time begins, the computation is aborted and you are asked to specify a time 1 hour later. If you specify a time during the double-valued period when the time drops back, the time defaults to standard and a rather sharper warning is printed. This makes one hour of real time inaccessible (!) unless you switch to greenwich time input with the `g` option and force the input time.

The routine to turn JD into calendar date is adapted from *Numerical Recipes in C* by Press *et al.* The routine to generate the JD from the date and time was adapted from a routine originally based on a recipe in the old *American Ephemeris*.

Sun and Moon.

The lunar positions used are computed from Jean Meeus' *Astronomical Formulae for Calculators*, Third Edition (1985, Willman-Bell: Richmond). The routine corrects the time argument to approximate TDT, because the moon moves quickly enough to make these small timing differences significant. Spot checks against the *Astronomical Almanac* indicate that the routine generates *geocentric* lunar positions good to better than a few seconds of time in RA and a fraction of a minute of arc in declination. A topocentric correction (from geocentric to observatory-centered, which can be $\sim 1°$!) is included, based on an ellipsoidal earth and the true elevation of the observatory. The topocentric correction appears to be somewhat more accurate than the lunar theory used.

As noted earlier, under the `=` command the program prints a notice if a solar or lunar eclipse is in progress. The solar eclipse state is found very directly by computing the topocentric angular radii of the sun and moon and comparing with their topocentric angular separation. The lunar eclipse calculation uses a simple geometrical model of the earth's shadow (taking into account the distance of the sun) at the moon's geocentric distance. Although the program does not generate eclipse timings directly, one can manually iterate to obtain times of eclipse contacts. This provides an exacting test of the lunar ephemerides and the topocentric correction. For solar eclipses, the timings agree to within about 1 minute with the definitive ephemerides (e. g., F. Espenak and J. Anderson, NASA reference publications Nos. 1301 and 1318, 1993); a 1 minute timing error implies $\sim 30''$ uncertainty in the moon's longitude. Lunar eclipse contacts are accurate to within about 5 minutes, with residual differences apparently due to the simple model used for the earth's shadow. Thus these programs should not be used for the most critical eclipse calculations.

I do not know over what range of dates the lunar ephemeris can be expected to work well, but it works nicely toward the end of the twentieth century.

The printed phases of the moon are based on Meeus' algorithms, which he claims are good to ± 2 minutes.

Explicitly-printed positions of the sun are also from algorithms derived from Jean Meeus *Astronomical Formulae for Calculators*. These positions are referred to the *mean* equinox of date. A topocentric correction is applied (which amount to at most 8.8 arcsec). Spot checks of the routine itself (modified for this purpose to show geocentric apparent rather than mean coordinates) gave agreement to a few arcseconds. Rise/set times are derived using the *Astronomical Almanac* low-precision formulae for the sun, which are advertised as good to about 0.01°.

If the observatory elevation above its horizon is specified as zero, the rising and setting times of the moon and sun are taken to be the times when the center of the object is 50 arcmin below the geometrical horizon. This is about the time of contact of the upper limb with the horizon, once refraction is taken into account. Variations in the apparent diameter of the sun and moon are ignored. If the observatory elevation above its horizon is non-zero, an approximate correction is added to the zenith angle at which rising and setting are reckoned; this is

$$\text{horizon correction (radians)} = \sqrt{\frac{2e}{R}},$$

where e is the observatory's elevation above its surroundings and R is the radius of the earth. In principle, a more accurate correction would simultaneously consider the effect of elevation on the refraction (although this doesn't make a great deal of difference – see B. E. Schaefer and W. Liller, 1990, PASP, 102, 796, Table 4). In extreme cases (Mauna Kea!) the horizon correction can affect rise/set times by some 10 minutes.

Spot checks Schaefer and Liller's table of *observed* times of sunset for Mauna Kea and Cerro Tololo gave (for the most part) agreement to within about a minute; refraction variations preclude more accurate prediction. The observatory elevation above its surroundings is used only in the rise/set computations; the barycentric corrections and the topocentric correction for the moon use the observatory's elevation above sea level, which is a separate parameter. Thus the elevation above the horizon may be adjusted to fit local circumstances. The NOAO Newsletter tables for Kitt Peak, for instance, have sometimes included a correction of several hundred meters (smaller than the 2 km elevation of the observatory), the purpose being to correct approximately for the fact that Kitt Peak is higher than most of the mountains which define its horizon.

At very high latitudes, where the moon and sun graze the horizon, the program is less accurate since it iterates the rising, setting, and twilight times until the altitude of the object is within 0.1° of the desired altitude. The rise and set algorithms are serviceable at circumpolar latitudes (see the section on geographical limitations below), but become increasingly unreliable within a couple of degrees of the poles, where they are useless (at the poles, the diurnal rotation does not affect the altitudes of objects)!

The *lunar sky brightness* contribution is estimated if the sun is well down (beneath −9 degrees altitude) *and* both the moon and the object are in the sky. This calculation follows K. Krisciunas and B. E. Schaefer (1991) PASP 103, 1033. The calculation will not be even roughly accurate unless the sky is quite clear; haze, cloud, or even volcanic aerosols high in the atmosphere can greatly affect the scattered moonlight! To the Krisciunas and Schaefer model I've added a correction for variations in the apparent size of the moon and an extremely crude model of the 'opposition effect', the surge of brightness just around full moon. This is modeled as a 35 per cent brightening at full moon, which tapers linearly in phase and goes to zero at 7 degrees from full moon. The code does check for *lunar eclipses*, but makes no attempt to account for their effect on the sky brightness. It does print a disclaimer if the moon is in eclipse. The brightness calculation assumes a zenith extinction of 0.172 mag in V, typical of the 2800 m level on Mauna Kea. Results are reported as equivalent V magnitude per square arcsecond; for comparison, the zenith night-sky brightness in a dark site is quite variable, but is very roughly 21.5 mag per square arcsec in V. *These estimates should be useful for planning purposes, but unlike some of the other results in this program they are unlikely to be very precise.*

Similar cautions apply to the zenith twilight brightness. This is based on a polynomial fit to a graph on p. 38 of A. and M. Meinel's lovely book *Sunsets, Twilights, and Evening Skies* (Cambridge: 1983). Comparison with measurements by E. V. Ashburn, *Journ Geophys Rsch*, v.57, p.85, 1952) shows that the fit provides a fair match to the observed twilight in the *blue* (4400 Å); the V band is about a magnitude fainter, and I should be a little fainter still. The zero point of this number – the dark night sky – is quite problematic, but the dependence on the sun's zenith distance should be reasonably accurate. Ashburn's data were taken from a California mountain site at an elevation of 1653 meters (5415 feet).

The Planets.

The purpose of the planetary calculations is not to give definitive positions (which are now derived from numerical integrations) but to give rough positions for planning purposes (*e. g.,* is Jupiter visible? Is it close to my object?). If you really need to point blindly exactly at a planet, get another program or consult the *Astronomical Almanac!*

The positions are computed using formulae from the 1992 *Astronomical Almanac* (p. E4). The input data are heterogeneous. For the planets through Mars, the program uses mean elements from the old *Explanatory Supplement* to the Nautical Almanac. These give very good results (usually less than 1 arcmin) for the inferior planets and satisfactory results (a few arcmin) for Mars. For the outer planets (Jupiter through Neptune), the input data are from Jean Meeus' *Astronomical Formulae for Calculators*, Third Editions (1985, Willman-Bell: Richmond). The outer planets have such large mutual attractions that satisfactory positions can only be had by including a fair number of perturbation terms; I have included the largest ones from Meeus' Chapter 24. The results are generally good to about 0.1 degree for Jupiter, and to a few tenths of a degree for Saturn, Uranus, and Neptune. For Pluto, I have simply adopted the osculating elements for 1992. These give very good positions for 1992, which slowly deteriorate the farther one gets from this date.

The planetary positions are used in two ways. They can be printed out in a table using the option `m`, which stands for "major planets" ("p" is already used for proper motion). More subtly, when one prints circumstances using the `=` command, the program computes the planetary positions and checks to see if your current RA and dec are within 3 degrees of any major planet. If they are, it warns you. The idea here is to avoid trying to observe some faint object with, say, Jupiter right next to it; the 3-degree tolerance was chosen as being about the radius of a Schmidt plate. For asteroids, you're on your own!

Geographical limitations.

The daylight savings time conventions used are limited to those which are coded. If you want to extend these to use at other sites you have to code the new convention into the program and assign it a numerical code; negative numbers refer to southern sites (daylight savings in, for instance December) and positive to northern sites. The routine to modify is called `find_dst_bounds`.

The algorithms used for rising and setting work at tropical and temperate latitudes, and have been retrofitted to work at very high latitude. As noted above, rise/set times are not as accurate at circumpolar latitudes as they are closer to the equator, and they are meaningless at the geographical poles. The code has not been tested exhaustively at very high latitudes, nor has it been tested at length in the southern or eastern hemispheres, but there are no reasons for expecting it won't work there. Problems with computation of moonrise, etc., which should arise only at extreme latitudes, are announced by a message reading

```
"Moonrise or -set calculation not converging!!".
```

These problems can arise because at very high latitudes, phenomena such as sunrise, twilight, and moonrise do not always occur. Thus the almanac section of the program tests that each of these phenomena are likely to occur before attempting to compute when they do occur. In this test, it uses the declination of the relevant body computed for local midnight; this can cause a mistake, especially for the moon, which can change declination quickly. This should seldom be important.

Precession.

The precession algorithm is coded directly from L. Taff's very useful book *Computational Spherical Astronomy* (Wiley). It uses a rotation matrix, which works correctly at the poles, and gives mean positions good to less than 1 arcsec in 50 years. It gives the same answers as the IRAF routine to this accuracy; also the set of test coordinates given by Smith et al. (1989, A. J. 97, 265) was reproduced to within 1 arcsec accuracy (except for proper motions near the poles). The present version of the program uses the IAU 1976 precession parameters. The program ignores such distinctions as that between B1950.0 and 1950 Jan 1; these are unimportant at this level of accuracy.

Note that this program does *not* compute the complete apparent place (including aberration, nutation, refraction, what you had for breakfast, etc.). If you need this, or if you need to transform coordinates at greatly sub-arcsecond accuracy (as in transforming astrometric catalogs onto each other), use another program! Also note that because of refinements in the reference frame, proper motions should in principle be transformed at the same time as coordinates (*e. g.*, in updating from B1950 to J2000); the current program ignores this.

Local Mean Sidereal Time.

Strictly speaking, the local sidereal time equals the hour angle of the vernal equinox; the *mean* sidereal time computed here is slightly different, because the effect of nutation on the location of the equinox is not included. This correction is called the 'equation of the equinoxes', which is tablulated in the *Astronomical Almanac*; it's generally less than 2 sec. The algorithm used here is based on formulae and procedures explained in the 1992 *Astronomical Almanac*, pp. B7 and L2. Tests for the longitude of Greenwich in 1992 give agreement with the Astronomical Almanac tables of *mean* sidereal time to within a few msec. Also, the IRAF routine gives the same answers to within 0.1 sec. However, strictly speaking this accuracy will obtain only if your input time is based on the correct type of UT; UTC (broadcast by WWV) is tied to atomic time and is corrected by whole seconds to agree with earth rotation. If your input is based on UTC (as all civil time is), the computed local mean sidereal time will be incorrect by UT − UTC, which is less than a second.

Parallactic Angle.

This quantity – the position angle of a great circle connecting the object to the zenith – is sometimes used for setting a spectrograph slit along the angle of atmospheric dispersion. Its use and importance are described by Alexei Filippenko (1982, PASP, 94, 715). Note that it generally is not important unless you are at least somewhat away from the zenith – Filippenko tabulates the dispersion. The angle is tricky to calculate because of subtleties in the choice of the root of an inverse trig function. A careful comparison with Filippenko's tables over a large range of hour angles and declinations gave complete agreement. Note that some applications (such as spectrograph slit angles) are indifferent to 180-degree rotations of this quantity; the antiparallel angle (parallactic ± 180) is also given, in square brackets.

Barycentric ('Heliocentric') Corrections

The algorithms used for the earth's orbit are derived from the solar ephemeris, which in turn is from Jean Meeus' *Astronomical Formulae for Calculators*, pp. 79*ff*. It uses an elliptical earth orbit and a few of the most important perturbations. The correction to the solar system barycenter is also included, using the same planetary calculations discussed earlier. The earth's diurnal rotation (assuming an ellipsoidal earth and including the observatory's elevation) is included in the velocity calculation, but the time-of-flight across the earth's radius ($\sim$ 0.02 sec) is not included. Meeus' solar theory does include a rough correction for the recoil of the earth due to the moon. Painstaking comparison with the tables of position and velocity of the earth in the *Astronomical Almanac* show that the time correction is generally good to $<$ 0.2 sec and the velocity to better than 5 m/s. The most demanding applications, such as analyses of pulsar timing and doppler-based searches for extrasolar planets, will require better accuracy, but this should be adequate for almost everyone else.

Galactic and Ecliptic Coordinates.

The galactic coordinates conform strictly to the IAU definition and agree closely with those computed by IRAF; they are based on a rotation matrix and do not suffer ambiguities due to the roots of inverse trig functions. The input coordinates are precessed to 1950 before being transformed to galactic, which introduces a slight uncertainty. If 1950 input coordinates are supplied, the only source of error should be double-precision roundoff! The ecliptic coordinates should be good to < 0.001 degrees.

BUGS and other ungraceful behavior.

I actually don't know of any real bugs (!), but the program can behave in a peculiar fashion given some inputs.

If you depart from the specified input formats, you can get peculiar behavior. Some error checking is done, and some prompting is given in some cases where really unexpected input is found, but these routines are less than perfect. It is difficult to crash the program or force it into an infinite loop.

The specification of times and dates is a little ungraceful (see the discussions of the `g` and `n` options above); the 'night date' option patches one potentially confusing condition with another. To some extent these confusions are inevitable; astronomers are forced to work at night, when dates change to suit the convenience of everyone else! The definition of JD, with its infernal half-step difference against UT dates, is a historical example of an ill-considered attempt to get around these difficulties.

In former versions, the day and date could in principle disagree within a very close tolerance of midnight. I believe I have eliminated the possibility of this by using the same truncation of the Julian date to derive the day and the date.

The conversion from local to UT is tricky around the time when daylight savings time changes to standard and *vice versa.* The behavior has been rationalized in recent versions, but it's still tricky. The hour when daylight time changes back to standard time is ambiguous – there is a default to standard time which may not be what the user wants. The user is warned if there might be a problem. Conversion from UT to local appears to be rigorously correct, so specifying times as UT when there is a problem should get around any difficulties.

After you type `g` to toggle between greenwich and local time, the time currently in effect changes to the value which is *numerically* the same in the new system, not the time which is actually equivalent. So if you are in the zone 7 hr west (Mountain), and you are using local time, and the time is 1991 Jul 7, 22 hr 0 mn 0 sec MST; and you type `g`, the time in force is now 1991 Jul 7, 22 hr 0 mn 0 sec *Universal time*, which is 7 hr *earlier.* To get the same actual time, you'd have to enter y 1991 7 8 and t 5 0 0. The program reminds the user that this is happening. Similar considerations apply to the `n` option.

Similarly, typing `e` to change the epoch assumed for the input coordinates doesn't precess any actual coordinates; it just changes how input will be interpreted.

On these points, it may be helpful to consider that the program doesn't actually calculate or convert anything until it's asked for output – the numbers you type in lie dormant until then. Thus the `g`, `n`, and `e` options control only the *interpretations* of your input parameters.

The twilight and rise/set times are slightly inaccurate at very high latitudes, since the object comes into the appropriate altitude at a grazing angle. Rise/set can be erroneously reported as not occuring at very high latitudes because the occurence of rise/set is judged using the position for local midnight, and it's possible in principle for the program to try to find a rise/set time which actually doesn't occur.

The correction used for the site elevation in the rise/set calculations is approximate. Note that it may or may not be appropriate to include altitude corrections for your site, based on the details of your local horizon.

Notes for Programmers.

The latest version (V4) of the `skycalc` calculator program allows the user to turn on a *log file*, so that one can save results without having to go through the rigamarole of creating an input file and redirecting output. I implemented this by replacing the appropriate calls to `printf` with a new routine, `oprntf`, which optionally writes to a file. Because `printf`, and hence `oprntf`, have variable numbers of arguments, I used the widely-available ANSI-standard framework (which involves the inclusion of `<stdarg.h>`; see Kernighan and Ritchie, 2nd edition, p. 155 `ff`). It turns out that the `cc` compiler on Sun workstations does not support this standard; the supposedly ANSI-standard version `acc` has a compiler bug affecting this particular feature (I'm indebted to Mike Fitz of NOAO for chasing this down). However, the Open Software Foundation's `gcc` compiles this correctly. I'd urge users of Sun machines to acquire this compiler; otherwise, a binary version is available at NOAO for Sun users.

It's not always appropriate for the user to have write permission. Accordingly it's possible to recompile the program with such permissions turned off; to do this, find the preprocesser definition

```
#define LOG_FILES_OK 1
```

and change the `1` to a `0`.

I encourage programmers to borrow from this code, but I caution against attempting major surgery on the code itself unless you study it for a while. There are some subtle issues involved which took me a while to get right. On the other hand, there are minor changes can be done safely and rather trivially.

If you find that the routines which depend on the system clock don't work, you can turn them off by finding the line

```
#define SYS_CLOCK_OK 1
```

and changing the `1` to any other number.

I've run into one curious issue regarding system clocks. At Lick Observatory, and perhaps others, there is a tradition of maintaining standard time all year for scientific purposes, even though civil timekeeping uses daylight savings time. It's tempting to implement this simply by turning off daylight savings, but then the `T` option doesn't work for half the year, because the computer's clock is generally set to civil time. I haven't yet coded in the complications needed to patch this.

To include a new site on the menu, modify the routine `load_site`. Follow the examples there. You can check that they are entered correctly by examining output from the `l` option in the program. Feel free to include new sites.

If your site uses an unsupported daylight savings time algorithm, include your option in the routine `find_dst_bounds`, using the current routine as a model. Note that in the labels for dst conventions, positive numbers refer to northern sites and negative to southern sites, for which the logic has to be reversed.

The code at this point is packaged as a single enormous source file, of over 5000 lines! Function declarations are in the 'old' K&R style, to keep Sun Microsystems compilers happy even in default. I also have a version in which the code is sliced into eight cross-referenced pieces, with ANSI-style function declarations. If you need this (say to compile on a PC), send me email.

2. A NIGHTTIME ASTRONOMICAL CALENDAR PROGRAM.

This program prints an astronomical calendar for a given year from a single site. The algorithms used are for the most part identical to those used for the circumstances calculator program described above, but the input and output are different. The purpose of the calendar is similar to that printed annually (up to 1994) in the NOAO newsletter for Kitt Peak; however, the format is designed to be easier to use, less error-prone, and to give rather more information than the NOAO calendar. (As of this writing, it looks likely that this program will in fact be used by NOAO for newsletters from 1995 onward!) The program can easily be adapted for other observatories. Again, this is a self-contained C program which should run gracefully on various computers; however, the same cautions as listed above apply.

The output has a wide format (122 characters). At the beginning, some information is printed along with prompts for interactive use (something which will probably seldom happen.) Then comes a page of information about the program and its accuracy, which is largely redundant with this document. Next follows a table of moon phases, with the times, given as local zone times, accurate to a few minutes. Next follow the results for each of the twelve months.

As of August 1993, an option is installed which formats the output so that it may be printed with the TeX * typesetting program. The output is minified so that it fits onto a page in the standard vertical orientation. Although it is no smaller than the tables in the NOAO Newsletter, you may wish to photocopy it sideways onto a page at greater scale. Or, another TeX option prints two months to a page, in which case you'll have a hard time magnifying it much more. Further details on the TeX option are given later. If TeX output is not selected, a formfeed character is inserted at the top of each page, so the output can simply be printed on a line printer with a new page for every month. I have also had good success porting the output to microcomputers and printing on a laserprinter in landscape mode.

At the head of each month is the year and month set off by asterisks. Also given is the site name, its longitude in *hours* minutes and seconds, its latitude in *degrees* and decimal minutes, and the standard time zone. After some other information, the user is reminded that the times listed (except for sidereal) are *local* times; the name of the zone is given. If daylight savings time is used, the user is reminded of this as well.

The rest of the calendar is organized with one night per line. Note that this choice is only sensible for nighttime astronomers, a large (but not all-inclusive) subset. Though the calendar works at circumpolar latitudes, this form of organization is not optimal during the "midnight sun" either! A detailed description of the tabulated quantities follows.

The first column gives the day and date, *for both evening and morning.* This should minimize errors in reading dates. A blank line appears between Saturday and Sunday nights.

The next column gives the JD at *local* midnight, rounded off to the nearest 0.1 d to avoid any ambiguity. The number given has the largest multiple of 10000 days (figured for the first of the month) subtracted away; thus JD 2451020.5 will be printed as 1020.5. If daylight savings is in use, the JD is the JD of daylight savings midnight.

* TeX is probably someone's trademark, or something.

The third column gives the Local *Mean* Sidereal Time (see the earlier discussion for the distinction between true and mean sidereal time) at local midnight; it is more accurate than the nearest-second accuracy which is tabulated. Again, if daylight savings is in effect, this is the JD at daylight-savings midnight.

The next four columns give respectively the times of sunset, evening (18-degree) twilight, morning twilight, and sunrise. Thus the columns are in the same sequence as the actual events, which seems natural. The twilight given is 18 degree ('astronomical') twilight, and the rise/set times given are when the center of the sun is 50 arcminutes below the horizon; this is roughly the time when the sun's upper limb touches the horizon, once refraction and the sun's angular diameter are taken into account. If the observatory's elevation above its surroundings are specified, the depression of the horizon is taken into account. Accuracy is as discussed above. If an event doesn't happen during a night (*e.g.*, twilight at high latitudes in summer), ellipses (...) are printed in the appropriate column. Note that the altitude of the observatory is not taken into account in the rise/set times; this can lead to disagreement by a few minutes with tables such as those in the *NOAO Newsletter*, which are sometimes computed assuming an elevation for the observatory.

The next two columns give a quantity I have found very useful, namely the sidereal times at evening and morning twilight. This defines the range of RA which is accessible on the meridian during the night.

The last five columns pertain to the *moon*.

The times of moonrise and moonset are given, provided they occur at night or within a boundary on either side. (The algorithm used here can give some trouble if the site is very far from the center of its assumed time zone.) Ellipses (...) are printed if the rise or set does not occur within the specified interval. Note that rise and set times, though they occur in successive columns, do *not* always occur successively in time, depending on the moon's phase. Rise/set times are again for 50 arcminutes below the horizon; variations in the moon's semidiameter are ignored. The moon ephemerides are based on accurate formulae from Jean Meeus' *Astronomical Formulae for Calculators.*

The next column gives the percentage of the moon's face which is illuminated. New, first quarter, full, and last quarter correspond approximately to values of 0, 50, 100, and 50 respectively for this quantity. If θ is the angle subtended by the sun and the moon at the observer, the quantity tabulated is

$$\text{Illuminated percentage} = 100. \times \frac{1}{2}(1 - \cos\theta).$$

Finally, the last two columns give the RA and dec of the moon at local midnight, whether or not the moon is up at that time. The position is topocentric. It is very useful to know the moon's position if you're trying to work around it.

Times in the Calendar program.

Note that the rise, set, and twilight times given in the calendar are for *the local time zone.* (The sidereal times are of course strictly local and have nothing to do with the time zone.) If you wish, daylight savings times can be listed; if you use this feature, a site-dependent prescription is used to select whether daylight or standard time is used. The switchover occurs at 2 AM on Sunday morning. This can in principle lead to an ambiguity around the time of time change (in the fall, it's 1:30 AM local time twice on the same night!), but you should be able to unravel this rare case from continuity with the preceding and following nights. Several conventions are available for the dates of the time change. The conventions coded of this writing are the USA convention (1st Sunday in April to last in October after 1986, last Sunday in April before 1986, more or less), the Chilean convention, and the Spanish convention. If you need another convention you'll have to add it to the source code, in the routine `find_dst_bounds`.

The header that appears on each page makes a note if daylight savings time is used. An asterisk is printed by the date on which the time is changed.

Running the calendar.

You will probably never want to actually run the calendar interactively; it takes a while to run, and it produces an output wider than most terminals. To run it, you'll want to use a background job, with output redirected to a file you can print on some wide device (*e.g.*, a laserwriter in landscape mode).

I describe below the input that the program will call for when run non-interactively. However, the program is also designed so that you can 'test-run' it interactively to reconnoiter the inputs it will require and the options available. The program first asks you to select a site, and prints a menu of 'canned' possibilities. You can select one of the 'canned' sites, or enter all the parameters for another site. The last input prompted for is the year for which to print the calendar; giving a negative year here exits the program.

Before producing the calendar, then, your first step should be to run the program interactively, mostly to be sure which 'canned sites' are available in your own version of the program. After that, you create a little procedure or job, with the output redirected into a file, to make the calendar itself. The exact format of this job will be dependent on your system, and on just what you want to do. However, the inputs you need will be system-independent. Here are some annotated examples; the text to the right is commentary, not to be put in the job itself.

Example 1 - for one of the 'canned sites.'.

```
k             (code for kitt peak, assumed to be
                              one of the "canned sites")
1991          (year for which calendar is to be run)
2             (do format for TeX printing, 2 months per page.)
```

Example 2 - for another site.

```
n               (new site - not one of the canned ones).
6 16 56         (WEST longitude, HOURS MINUTES SECONDS.)
44 44 42        (latitude, DEGREES MINUTES SECONDS).
0               (site elevation, in meters, above effective horizon)
6               (standard time zone, hours WEST of Greenwich)
USND Hoople     (Site name; terminate with carriage return).
Central         (Time zone name, terminated with carriage return).
1               (use daylight savings time,
                          USA post-1986 prescription).
0               (don't use TeX on output -- give 1 or 2 for TeX).
1991            (year)
```

Naturally, you should be especially careful about your site parameters; anyone entering a new site in the source code should be downright compulsive, since many people may depend on the accuracy. The user should check the output to be sure the parameters repeated are correct; the latitude, longitude, etc. are printed at the top of every month's page.

Examples of how to run the program in background.

I'll show how to do this using UNIX or VMS systems (note that these are trademark names). This is not of complete generality, but covers the bases for most users.

On a UNIX system, if you've named your executable task `calendar`, you'd edit a file called `inputs` containing the input data, just as above. Then type

```
calendar < inputs > hoople_1991 &
```

which could be paraphrased as "run the calendar program, taking input from (`<`) the file named `inputs`, directing output to (`>`) a file named `hoople_1991`, and do it in background (`&`)".

On a VMS system, you would edit a command file – let's call it `CALDRIVE.COM` – which would have the first line

```
$ run calendar
```

with subsequent input on successive lines without dollar signs at the front (again, just as above). Then you'd type

```
@CALDRIVE/OUT=HOOPLE_1991.LIS
```

to run the command file and direct the outfile to a file called `HOOPLE_1991.LIS`.

More on the TeX option.

The TeX output is based on a 'Dirty Trick' given by Donald Knuth in *The TeXbook* on page 382; his macro `verbatim` simply prints a section of text using the `tt` font, which has a fixed character width. If you select the TeX option, you'll have to edit your redirected output file to remove the 'fossil prompts' which come at the beginning; look for the line marked `CUT HERE`. The edited file *should* then set up and print normally. If you used option 1, you'll get one month on each page; if you used option 2, you'll get two months on one page. You'll also get a cover page and the moon-phase table.

Note that there are several parameters right at the start of the TeX file which may need to be 'tweaked' to your local printer. They are `\magnification 835`, `\hsize 7.6 truein`, and `\hoffset -0.7 truein`. These parameter values simply make the very wide output as large as possible on our local system. Because TeX defaults to a magnification of 1000, the value 835 makes the print rather small (though not significantly smaller than the annual NOAO newsletter tables). This can be remedied, if need be, with a little photocopier work. You may wish to make the `\baselineskip` a little larger if you find the lines to be too crowded vertically; increasing it to 12 spreads things out a lot.

Sample Output from the Calendar Program.

The following output came directly from a run of the program using the input quantities given in the example above. It should be used to check the results in a new site.

Because the output is so wide, I've added carriage returns; in the main body of the calendar, they are consistently right after the columns relating to the sun. This has a terrible effect on legibility, but fits it on the page ...

```
                                *****  1991 JANUARY *****

Calendar for Univ. South. North Dakota at Hoople, west longitude
(h.m.s) = 6 16 56, latitude (d.m) =  44 44.7
Note that each line lists events of one night, spanning two calendar
dates.  Rise/set times are given
in Central time (  6 hr W), uncorrected for elevation, DAYLIGHT time
used, * shows night clocks are reset.
Moon coords. and illum. are for local midnight, even if moon is down.
Program: John Thorstensen, Dartmouth College.

  Date (eve/morn)      JDmid    LMSTmidn   ---------- Sun: ---------
LST twilight:  ------------- Moon: --------------
  (1991 at start)    (-2440000)            set  twi.end twi.beg rise
    eve    morn    rise   set  %illum   RA       Dec

Tue Jan 01/Wed Jan 02  8258.8    6 28 34   16 47  18 33   6 09  7 54
    1 00  12 38   18 08  .....    97   8 17.7  18 56
Wed Jan 02/Thu Jan 03  8259.8    6 32 31   16 48  18 33   6 09  7 54
    1 05  12 42   19 29  .....    91   9 15.5  13 45
```

```
Thu Jan 03/Fri Jan 04  8260.8    6 36 28   16 49  18 34   6 09  7 54
    1 10  12 46   20 47  .....    83  10 08.9   7 58
```

3. CAUTIONS APPLYING TO BOTH PROGRAMS.

When these codes are ported to a new system, the results should be checked carefully for accuracy. The sample output in this document should be reproduced correctly. The user assumes responsibility for the correct operation of the programs and the sensible interpretation of their results. The user's attention is drawn to the known limitations of the algorithms documented above.

While the programs have been tested carefully, with the results given above, the author makes no guarantee that this level of accuracy will obtain in all circumstances on all machines. I explicitly disavow any responsibility, express or implied, for damages resulting from use of the program. Output from this program should never be used as evidence in a court of law, or to make decisions which might cause bodily harm if the results weren't right.

Miscellany

The source code is usually stored as monolithic files, containing all the subroutines as well as the main programs. At least one user has reported that the size causes difficulty on some personal computers (e. g., Macintosh). To alleviate this I have included function prototypes (commented out) in the most recent version of the almanac program; these should facilitate breaking the code into smaller pieces. This list is not necessarily up-do-date (I have other duties to attend to). While earlier versions of the almanac program returned structure values from functions, the current version does not, because some compilers complain if returned structures are at all sizeable.

Maintainance.

If you find a real problem, not due to your local machine and not documented above, write

John Thorstensen

Dept. of Physics and Astronomy

Dartmouth College

Hanover, NH 03755

John.Thorstensen@dartmouth.edu