Date Algorithms

By Peter Baum
Onset, MA 02558-1255

Aesir Research
Version 1 of this document is also available as HTML online at archive.org.

Version: 5
Previously Modified: April 28, 2017; Version 2
Tuesday, March 3, 2020; Version 3
Sunday, October 18, 2020; Version 4
Last Modified: Wednesday, October 21, 2020
See Revision List for more details about version differences.

Table of Contents

0. Introduction

The information contained in the sections listed below is a guide to computer programmers interested in implementing date algorithms in a variety of languages and on a wide range of hardware platforms. Although all rights are reserved under the copyright laws, programmers may implement the algorithms without permission. However, I would ask that if this is done for any of the algorithms that are not known to be fully documented elsewhere, a reference be made to this document. Doing so will assure full and consistent documentation, minimize the propagation of any errors that might be discovered, and provide a central location for any improvements that are made in the future.

Various calendar dates (Julian, Gregorian, and Rata Die) are used and produced by the algorithms that are presented but no attempt has been made to account for various problems associated with the general use of historical dates. For example, 90 days were inserted into the calendar by Julius Caesar in -45 and following Caesar's death, the leap year rule was misapplied until corrected by Emperor Augustus who omitted leap years from years -8 through +4. Various locations used different calendars throughout the Middle Ages and even within one locality, different calendars were often used for dating ecclesiastical records, fiscal transactions, and personal correspondence. None of these complications are accounted for by these algorithms. The following is assumed:

- The year before the year 1 is 0 and the year before 0 is -1, et cetera. The year 0 corresponds to 1 B.C.
- Julian dates may be applied both before the calendar was invented and after it was replaced as long as it is clear that the Julian leap year rules are being applied, usually by calling the date a Julian date.
- Gregorian dates may be applied before the Gregorian calendar was invented as long as it is clear that the Gregorian calendar leap year rules are being applied, usually by calling it a Gregorian date.

1. Calendars and Dates

DEFINITION: The **Julian calendar** is the calendar established by Julius Caesar in 46 B.C. (or the year -45, if the convention of a year zero and negative numbers is used). In this calendar, the year is fixed at 365 days except for every 4th year when it has 366 days. Alternatively, the year can be considered to be 365.25 days long but fractions are ignored until they accumulate to make a full day. Dates in the Julian Calendar are called **Julian dates** and are not to be confused with Julian Day numbers.

DEFINITION: The **Gregorian calendar** is the reformed Julian calendar with the year fixed at 365 days except for leap years which contain 366 days. Leap years are years that are exactly divisible by 4 except for century years which must also be divisible by 400. This is the calendar in common use today. Dates in the Gregorian calendar are called **Gregorian dates**.

DEFINITION: The **Julian proleptic calendar** is a calendar formed by applying the rules of the Julian calendar back in time, even before the Julian calendar was invented.

DEFINITION: The **Rata Die** number is a date specified as the number of days relative to a base date of December 31 of the year 0. Thus January 1 of the year 1 is Rata Die day 1.

DEFINITION: The **Julian Day Number**, **Julian Day**, or **JD** of a particular instant of time is the number of days and fractions of a day since 12 hours Universal Time (Greenwich mean noon) on January 1 of the year -4712, where the year is given in the Julian proleptic calendar. The idea of using this reference date was originally proposed by Joseph Scalizer in 1582 to count years but it was modified by 19th century astronomers to count days. One could have equivalently defined the reference time to be noon of November 24, -4713 if were understood that Gregorian calendar rules were applied. Julian days are Julian Day Numbers and are not to be confused with Julian dates.

If the JD number is expressed as an integer, it represents the number of whole days since the reference instant to noon of that day. Table 1 gives some examples.

Table 1

Gregorian Date	Time	JD number
December 31, 1979	noon	2444239.0
January 1, 1980	start of the day (just after midnight)	2444239.5
January 1, 1980	noon	2444240.0
January 1, 1980	no time specified	2444240
January 1, 1980	midnight commencing January 2	2444240.5

A Julian date is a date in the Julian calendar. Similarly, a Gregorian date is a date in the Gregorian calendar. Table 2 shows how Julian dates and Gregorian dates relate to the Julian Day Number. The year, month, and day given in the table can represent either a Julian date or a Gregorian Date. If it is considered a Julian date, the Julian Day Number is given in the column with heading "JD for Julian date." If the date is considered a Gregorian date, the Julian Day Number is given in the column with heading "JD for Gregorian date."

Numbers in the "Day" column represent days that begin at midnight. This contrasts with Julian Day Numbers which start at noon and continue up until noon of the following day. Thus a day number such as 24.5 represents noon and has a JD number that ends in zero tenths.

Table 2					
Year	**Month**	**Day**	**JD for Julian date**	**JD for Gregorian date**	**Comment**
-4713	11	24.0	-38.5	-0.5	middle of JD before Gregorian reference
-4713	11	24.5	-38.0	0.0	Gregorian date of JD reference
-4713	11	25.0	-37.5	0.5	middle of JD reference day (midnight)
-4712	1	1.0	-0.5	37.5	middle of day before JD reference
-4712	1	1.5	0.0	38.0	Julian date of JD reference
-4712	1	2.0	0.5	38.5	middle of JD reference day
0	1	1.0	1721057.5	1721059.5	The first day of 1 B.C.
0	2	29.0	1721116.5	1721118.5	Rata Die = -306
0	3	1.0	1721117.5	1721119.5	Rata Die = -305
0	12	31.0	1721422.5	1721424.5	Rata Die = 0
1	1	1.0	1721423.5	1721425.5	Rata Die = 1
1582	10	4.0	2299159.5	2299149.5	last day before Gregorian reform
1582	10	15.0	2299170.5	2299160.5	first day of Gregorian reform
1840	12	31.0	2393482.5	2393470.5	M programming language reference
1858	11	17.0	2400012.5	2400000.5	Modified Julian Day 0.0
1900	1	1.0	2415032.5	2415020.5	
1901	1	1.0	2415398.5	2415385.5	start of the 20th century
1970	1	1.0	2440600.5	2440587.5	Unix reference
1980	1	1.0	2444252.5	2444239.5	PC (DOS) reference

The Julian Day Number is defined relative to the Julian proleptic calendar so that from noon of October 4, 1582, the last day in the Julian Calendar, there are 2299160.0 days to noon of the Julian Day Number reference time. This count is a result of considering dates that have leap years every 4 years. Noon of the day following Thursday, October 4, 1582, (Julian Calendar) is the first day of the Gregorian reform (designated as Friday, October 15, 1582, in the Gregorian calendar) and is JD 2299161.0. Normally from this date, the Gregorian calendar is used and the Gregorian calendar rules for leap year apply. It is, of course, possible to continue the use the Julian Calendar after the Gregorian reform or the Gregorian Calendar before the Gregorian reform. Using the Julian Day Number avoids this possible confusion but the system is not in common, everyday use by the general public.

2. Gregorian or Julian date to Julian Day Number

2.1 Gregorian or Julian date to Julian Day Number - Algorithm Development

Algorithm development begins by choosing a base year. Converting from a Gregorian calendar date implies that the base year is relative to the year zero. If the given date is relative to some particular year in the Gregorian calendar, then that date becomes our base year.

The month number and year number will be converted so that January and February are considered months 13 and 14 rather than months 1 and 2. January and February of year Y are converted so that they occur in the year Y - 1. In other words, a system is used where the first month of the year is March (designated as month 3) and the last month of the year is February of the following year and designated as month 14. The purpose of this scheme is to cause the extra day in a leap year to occur at the end of the "year." As will be seen, this simplifies the calculation of the number of leap years from year zero to a particular date and it simplifies the function which relates month number to the number of days in the preceding months.

Since the base year is March 1 of year 0, the Julian day number is determined relative to that date. From table 2, it can be seen that March 1.0 of year 0 is JD 1721119.5, assuming the date is in the Gregorian calendar. Thus March 1.5 of year 0 is JD 1721120.0 and is the Julian Day Number if a date of March 1 of year 0 is specified. Thus 1721118.5 must be added to the day number (of one) to get the correct JD. Note, however, that dates in the Gregorian calendar are assumed even though the base date is before the Gregorian calendar was invented. This works fine as long as the dates input to the algorithm are Gregorian calendar dates or dates in the Gregorian proleptic calendar.

Next, the days in the year are accounted for. This is easily done using the following function:

Y*365 +[Y/4] -[Y/100] + [Y/400]

This expression directly encodes the rule that there are 365 days in a year plus an extra day for leap years, where leap years occur every 4 years except for century years unless they are also divisible by 400. Here [x] is the **greatest integer function**, that is, [x] is the greatest integer that is not greater than x. Thus [2]=2, [2.5]=2, [0]=0, [-1.5]=-2, and [-3]=-3. For positive numbers, the greatest integer is the same as a function which truncates any fractional part of a number.

Note that although 0 is a leap year, the number of days relative to March 1 of the year zero are being considered. Relative to March 1 of the year zero, the first leap year is year 4 and February 29, 4, is part of the year beginning March 1, 3. The expression - [Y/100] eliminates all century years as leap years while [Y/400] adds back century years that are divisible by 400.

Finally, the number of days in the months prior to the specified month are accounted for. The following gives the range and domain of the function explicitly:

month	days in previous months
3	0
4	31
5	61
6	92
7	122
8	153
9	184
10	214
11	245
12	275
13	306
14	337

Since, all the months except February are either 30 or 31 days long, one might suspect that a linear approximation can be used and the fractional part of the value removed to generate a convenient form of the function. One can not be certain that this will work on a particular function before it is examined in more detail, but, as will be shown, the function given in the table can be successfully generated in this way.

The function will take the form:
FIX(M*X+B)
where
X = month number,
M = the slope of the line,
B= the Y intercept of the line, and
FIX() is the truncation function.

Now since FIX() removes the fraction, the values of MX+B that produce the same FIX function value are values of MX + B from some integer, say Y, up to, but not including, the value Y+1. The values for each Y can be graphed, but because of the scale, it is difficult to see precisely how the line might fit through all the function values or whether or not it is even possible to do so. However, it is not difficult to write a computer program that determines what equations can pass through these intervals. Doing so shows that the line with maximum slope goes from point (4,31) to (12,276), has slope= 30.625 and Y intercept=-91.5. The line with minimum slope goes from point (14,337) to (7,123), has a slope= 30.57142857142857 and Y intercept=-91.0000000000000. These two lines cross at (9.333333333333067,194.3333333333252) and I call this the pivot point, since there will be lines within the specified intervals that go through this point and take on any of the slopes from 30.57142857142857 to 30.625. Given slope M, the values 194.3333333333252 - M * 9.333333333333067 can be used as the Y intercept. However, there will usually be many other Y intercept values that work as well.

From a computational point of view, one probably notices that a slope of 30.6 contains a minimum number of digits and should not be too difficult to produce using integer arithmetic. If for slope = 30.6, the line goes through the pivot, the Y intercept is -91.26666666666669 but there are lots of other Y intercept values that work as well. These Y intercept values can range from -91.4 to -91.2, although the value of -91.2 cannot actually be

taken since the line would go through a couple of points that are integers and the truncation function would not produce the desired result. Thus the function

FIX(30.6*X-91.4)

can be used to produce our desired function values. The function can also be converted to other forms for convenience. For example

FIX(30.6*X-91.4)
= -122 + FIX(30.6*X-91.4+122)
= -122 + FIX(30.6*X+30.6)
=-122 + FIX(30.6*(X+1))

FIX(30.6*X-91.4) is equivalent to (153*m-457)\5 since 153/5=30.6 and -457/5=-91.4.

Other coefficients for our generating function are also possible, and they can be searched for using the bounds described above. Suppose, for example, that the function is to have the form (A*X+B)\32 so that shifts can be used in the binary representation to implement integer division. Since the slope must range from 30.57142857142857 to 30.625, A/32 must also be in this range. This implies that A must range from 30.57142857142857 * 32 = 978.28571428571424 to 30.625*32= 980.0. The integer 979 is an obvious choice. This means the slope is 979.0/32=30.59375. This slope implies that the Y intercept is in the range -91.15625 to -91.3125. Thus B must be in the range of -91.15625*32= -2,917.0 to -91.3125*32=-2,922, where the larger of the intercepts (-2,917) is excluded as before. Thus the range choice for B is -2,922 through -2918 inclusive. Thus the alternative formula: (979*m-2918)\32 can be used to generate the required function. Here "\" is a shift which implements integer division with truncation rather than the greatest integer function. This is possible because variable "m" has a value of 3 or greater so that the numerator is always positive.

2.2 Julian Day Number - Algorithm Forms

2.2.1 Gregorian Date to Julian Day number

The previous algorithm development section justifies the following general algorithm for converting a Gregorian Date into a Julian Day Number. In this description:

[x] = the greatest integer that does not exceed x. For example, [-1.5]=-2. This is sometimes called the floor function (for example in C/C++).

INT(x) = [x] NOTE: some computer languages have a different definition.

FIX(x) = the number x without its fraction. For example, FIX(-1.5)=-1

x\y = FIX(x/y)

STEP 1 Calculate the value of Z using **ONE** of the following methods:

1. Z = Y + (M-14)\12
2. IF M<3 THEN Z=Y-1 ELSE Z=Y
3. On some computers, the month and year can be placed in memory locations so that a subtraction of the month will carry into the year when the month is January or February.

STEP 2 Calculate the value of F using **ONE** of the following methods:

1. F is the value of a vector indexed by M. This vector has values 306, 337, 0, 31, 61, 92, 122, 153, 184, 214, 245, 275. This method is usually the fastest method and often takes less computer memory to implement.
2. IF M<3 THEN M=M+12
   F = (153*M-457)\5
3. IF M<3 THEN M=M+12
   F = (979*M-2918)\32
   This allows a shift to carry out the division.
4. F = (979*(M-12*((M-14)\12))-2918)\32
   This eliminates the need for an IF test and carries out the division using a shift.
5. If real arithmetic is desired,
   IF M<3 THEN M=M+12
   F=FIX(30.6*M-91.4) = -122+FIX(30.6*(M+1))
   can be used although real arithmetic is usually slower and takes more code space. Other coefficients are possible as described in the previous section. Note that since M is 3 or greater, 30.6*M-91.4 is always positive and either FIX or the greatest integer function can be used.

STEP 3 The Julian Day Number for a Gregorian date is

D+F+365*Z+[Z/4]-[Z/100]+[Z/400] + 1721118.5

This expression is valid for any Gregorian date, even proleptic Gregorian dates including those for negative years.

For example, the following sequence can be used to calculate the Julian Day Number from a Gregorian date:

```
IF M < 3 THEN
    M = M + 12
    Y=Y-1
END IF
JD = D + (153 * M - 457) \ 5 + 365 * Y + [Y / 4] - [Y / 100] + [Y / 400] + 1721118.5
```

2.2.2 Julian Date to Julian Day number

There are two modifications of the Gregorian date to Julian Day Number algorithm that needs to be made in order to produce a Julian date to Julian Day Number algorithm. First, the base date of March 1 of the year 0 as a Julian date is 1721117.5 so our constant is 1721116.5 rather than 1721118.5. Thus, for March 1, 0 (Julian date), our day number is 1, which, when added to our constant of 1721116.5 producing the desired JD of 1721117.5. The second modification is to count leap years every 4 years and not exclude the century years that are not divisible by 400. Thus our account of leap years is simply 365 * Y + [Y / 4] which is sometimes expressed as [365*Y+[Y/4]] = [365.25*Y]. Thus our Julian Day Number for a Julian date is

D+F+365*Z+[Z/4] + 1721116.5

where the values of D, F, and Z are as before. For example, the following sequence can be used to calculate the Julian Day Number from a Julian date:

```
IF M < 3 THEN
    M = M + 12
    Y=Y-1
END IF
JD = D + (153 * M - 457) \ 5 + 365 * Y + [Y / 4] + 1721116.5
```

2.3 Gregorian date to Julian Day Number - Implementation

2.3.0.1 Choices to be made

The following decisions have to be made regarding any implementation:

- Are dates specified in the Gregorian or Julian Calendar?
- What range of dates are to be supported?
- What number word size will be used?
- Is the greatest integer function available or is only the truncation function?
- Is real arithmetic available and required?
- Is conditional execution such as "IF/THEN" supported?

2.3.0.2 Notation

[x] = the greatest integer that does not exceed x. For example, [-1.5]=-2. This is sometimes called the floor function (for example in C/C++).

INT(x) = [x] NOTE: some computer languages have a different definition.

FIX(x) = the number x without its fraction. For example, FIX(-1.5)=-1

x\y = FIX(x/y)

2.3.1 Gregorian Date to Julian Day Number Implementation Examples

1. For a Gregorian Date specified as day D (a real number), month M (an integer with January = 1), and year Y (any integer), the Julian Day Number JD can be calculated as follows:

   ```
   IF M < 3 THEN
       M = M + 12
       Y=Y-1
   END IF
   JD = D + (153 * M - 457) \ 5 + 365 * Y + [Y / 4] - [Y / 100] + [Y / 400] + 1721118.5
   ```

   COMMENTS:

   - The range of years that are supported is only limited by the word size of the implementation.
   - Although (153 * M - 457) \ 5 is an elegant expression, an indexed array is usually a faster implementation.
   - Some languages only implement the greatest integer function for a real argument.

2. QBASIC Implementation as a subroutine:

   ```
   SUB jdg (jd AS DOUBLE, y1 AS LONG, m1 AS INTEGER, d AS DOUBLE)
   y& = y1 'don't modify year input parameter
   m% = m1 'don't modify month input parameter
   IF m% < 3 THEN 'months are January or February
   ```

```
    m% = m% + 12 'January=13, February=14
    y& = y& - 1 'of the previous year
END IF 'months are January or February
jd=d#+(153*m%-457)\5+INT(365.25#*y&)-INT(y&*.01#)+INT(y&*.0025#)+1721118.5#
END SUB
```

3. In *Astronomical Algorithms*,1991, page 61, Jean Meeus gives the algorithm (essentially) as

```
IF M < 3 THEN
    M = M + 12
    Y = Y - 1
END IF
JD=FIX(365.25(Y+4716))+FIX(30.6001(m+1))+D+2-FIX(Y/100)+FIX(FIX(Y/100)/4)-1524.5
```

Since 365.25*4716 = 1,722,519.0,
and 30.6001(m+1)) = FIX(30.6001*m - 91.4) + 122,
and FIX(FIX(y/100)/4) = FIX(Y/400),

and since 1,722,519.0 + 122 + 2 - 1524.5 = 1721118.5,
this is equivalent to our derived algorithm provided the years are positive.

COMMENTS:

- This algorithm is only valid for non-negative Y. It can be made suitable for negative Y by replacing the FIX function with the greatest integer function.
- The constant 4716 was introduced in the process of creating an expression that would allow for the JD to be calculated for negative Julian Dates. It unnecessarily complicates the algorithm for Gregorian dates.
- The constant 30.6001 is used instead of 30.6 because of concern for rounding errors. The previous derivation of the constant 30.6 shows that this is an unwarranted concern.
- There is nothing gained by using
  FIX(FIX(Y/100)/4) rather than FIX(Y/400)
  or equivalently,
  A=FIX(Y/100)
  and using -A+FIX(A/4) for the value of - [Y / 100] + [Y / 400]
  unless A/4 is performed using a shift operation.

4. For dates in the Gregorian calendar at noon, *The Explanatory Supplement to the Astronomical Almanac*,1992, page 604, gives the value for the Julian Day Number for a Gregorian Date after November 23, -4713 as:

```
JD=(1461*(Y+4800+(M-14)\12))\4+(367*(M-2-12*((M-14)\12)))\12-(3*((Y+4900+(M-
14)\12)\100))\4+D-32075
```

Here the expression:
(M - 14)\12
is used to implement the conditional change of Y and M that can be more clearly expressed as:

```
IF M < 3 THEN
    M = M + 12
    Y = Y - 1
END IF
```

With this change in the values of M and Y,
JD=(1461*(Y+4800))\4+(367*(M-2))\12-(3*((Y+4900)\100))\4+D-32075

Assume Y is non-negative (or use INT instead of FIX in the following argument). The following identities can be shown to hold:

1. (1461*(Y+4800))\4 = FIX(365.25*Y) + 1753200
2. 367*(M-2))\12 = FIX(30.583333333*M - 91.166666666) + 30
3. -(3*((Y+4900)\100))\4 = -FIX(Y/100) + FIX(Y/4) - 36

therefore the formula can be rewritten as

JD = FIX(365.25*Y) + FIX(30.583333333*M - 91.166666666) - FIX(Y/100) + FIX(Y/4) + D - 32075 - 36 + 30 + 1753200

= FIX(365.25*Y) + FIX(30.583333333*M - 91.166666666) - FIX(Y/100) + FIX(Y/4) + D + 1721119

which was shown in the development section to be the JD provided that the date specification is for noon and the years are non-negative.

COMMENTS:

- The algorithm is not valid for Gregorian years less than -4800.
- The reference time is for noon of the specified day. Thus the JD is from noon of the specified day until noon of the following day.
- The constants 4800 and 4900 were introduced so that the FIX function produces the correct result for negative Gregorian dates as small as -4800.
- If INT is available or the years are non-negative,
  using INT(365.25*Y) or 365*Y + INT(Y/4) in the algorithm is a better choice than
  (1461*(Y + 4800))\4
  since the range of years is greater.
- If INT is available or the years are non-negative,
  -INT(Y/100) + INT(Y/4) is a better choice than -(3*((Y+4900)\100))\4

5. Python Code Example for Gregorian to JDN
   WARNING Note that

- the Python // implements the greatest integer function, instead of the INT() used in this document and
- the Python int() implements the FIX() notation used in this document.

```
def GregorianToJDN(M,D,Y):
   """Gregorian to Julian Day Number Calculation"""
   if M<3:
      M+=12
      Y-=1
   return(D + int((153 * M - 457) / 5) + 365 * Y + Y // 4 - Y // 100 + Y // 400 + 1721118.5)
```

2.3.2 Julian date to Julian Day Number Implementation Examples

1. For a Julian date specified as day D (a real number), month M (an integer with January = 1), and year Y (any integer), the Julian Day Number JD can be calculated as follows:

```
IF M < 3 THEN
    M = M + 12
    Y=Y-1
END IF
JD = D + (153 * M - 457) \ 5 + 365 * Y + [Y / 4] + 1721116.5
```

   COMMENTS:

   - The range of years that are supported is only limited by the word size of the implementation.
   - Although (153 * M - 457) \ 5 is an elegant expression, an indexed array is usually a faster implementation.
   - Some languages only implement the greatest integer function for a real argument.

2. QBASIC Implementation as a subroutine:

```
SUB jdj (jd AS DOUBLE, y1 AS LONG, m1 AS INTEGER, d AS DOUBLE)
'***WARNING**** Input Date is Julian, NOT Gregorian
y& = y1 'don't modify year input parameter
m% = m1 'don't modify month input parameter
IF m% < 3 THEN 'months are January or February
    m% = m% + 12 'January=13, February=14
    y& = y& - 1 'of the previous year
END IF 'months are January or February
jd=d#+(153*m%-457)\5+INT(365.25#*y&)+1721116.5#
END SUB
```

3. In *Astronomical Algorithms*,1991, page 61, Jean Meeus gives the algorithm for a Julian Day number from a Julian date (essentially) as

```
IF M < 3 THEN
    M = M + 12
```

```
    Y = Y - 1
END IF
JD = FIX(365.25(Y + 4716)) + FIX(30.6001(M + 1)) + D - 1524.5
```

Since 365.25*4716 = 1,722,519.0,
and 30.6001(M + 1)) = FIX(30.6001*M - 91.4) + 122,

and since 1,722,519.0 + 122 -1524.5 = 1721116.5,
this is equivalent to our derived algorithm provided the years are not less than -4716. This is assured if the JD is non-negative since JD = 0.0 occurred at noon on January 1, -4712 in the Julian Calendar.

COMMENTS:

- This algorithm is only valid for Y not less than -4716 (after any possible adjustments for January and February). The intent of the form with (Y+4716) is to assure that the argument of the FIX function is always positive for the intended range of Y values. The algorithm can be made suitable for any negative Y by replacing the FIX function in the expression FIX(365.25(Y + 4716)) with the greatest integer function. Once this is done, the complication of using the constant 4716 can be dispensed with.
- The constant 30.6001 is used instead of 30.6 because of concern for rounding errors. The previous derivation of the constant 30.6 shows that this is an unwarranted concern.

3. Julian Day Number to Gregorian date

3.1 Algorithm Development

The basic strategy is to:

1. Convert the Julian Day Number to the number of days from some base date.
2. Determine the number of years from the base date.
3. Calculate the number of remaining days once the years are accounted for.
4. Determine the month number represented by the remaining days.
5. Calculate the number of remaining days once the months are accounted for.

3.1.1 Choosing a Base Date

Algorithm development begins by choosing a base date that our day count will be relative to instead of the less convenient Julian Day Number 0. Since conversion will be to a Gregorian date, this base date will be in the Gregorian year zero.

The Julian Day Number is converted to the number of days from a base date of March 1 in the year zero. This creates a calendar with January and February as months 13 and 14 rather than months 1 and 2. The purpose of this scheme is to cause any extra day in a leap year to occur at the end of the "year." As will be seen, this simplifies the calculation of the number of leap years from year zero to a particular date and it simplifies the function which relates the number of days in the preceding months to the month number.

Let MOD be the modulo function. Gregorian year Y is a leap year if Y MOD 4 = 0 and if Y MOD 100 = 0 then it is also required that Y MOD 400 = 0. In our year starting on March 1, the days in any January 1 based year span two different March 1 based year numbers. For example, if February 29 is in year Y for a year beginning January 1, it is in year Y-1 for a year beginning March 1. Similarly, to see if there is an extra day due to a leap year in year Y of a calendar beginning March 1, then our Gregorian leap year test must be applied to year Y+1.

From Table 2, it can be seen that in order to convert our Julian Day Number to the number of days since March 1 of the year zero in the Gregorian calendar, 1721119.5 must be subtracted. If day 1 is to be **on** March 1 of the year zero in the Gregorian calendar, 1721118.5 would be subtracted.

3.1.2 Determining the Year

With the number of days now relative to our base of March 1 of the year 0, the number of full years within the given time span in days is determined. Given year Y, the number of days to account for before the 1st day of that year depends upon the way leap years are calculated. For a Gregorian date, the number of days accounted for by the full years is 365*Y+[Y/4] -[Y/100]+[Y/400] where [x] is the greatest integer function. Here the extra leap year days are added because the date for a calendar beginning March 1 places February 29 in the preceding year.

3.1.2.1 Average days per year approximation

The number of days accounted for by a span of Y years, although equal to 365*Y+[Y/4] -[Y/100]+[Y/400], is approximately equal to 365*Y+Y/4 -Y/100+Y/400 = Y*(365 + 1/4 - 1/100 +1/400) = Y*365.2425 since the

greatest integer function changes the value of its argument by less than 1. If D is the day number of year Y with values 1 through 365 (or 366 if a leap year), then the task is to find Y given

365*Y+[Y/4] -[Y/100]+[Y/400] + D days.

Y must then be approximately

INT((365*Y+[Y/4] -[Y/100]+[Y/400] + D)/365.2425)

Examining the error from this approximation:

INT((365*Y + [Y/4] - [Y/100] + [Y/400] + D)/365.2425)
=INT((365/365.2425)*Y + ([Y/4] - [Y/100] + [Y/400] + D)/365.2425)
=INT((365/365.2425)*Y + ([Y/4] - [Y/100] + [Y/400] + D)/365.2425 + (Y/4 - Y/100 + Y/400)/365.2425 - (Y/4 - Y/100 + Y/400)/365.2425)
=INT((365*Y)/365.2425 + ([Y/4] - [Y/100] + [Y/400] + D)/365.2425 + (Y/4 - Y/100 + Y/400)/365.2425 - (Y/4 - Y/100 + Y/400)/365.2425)
=INT((365*Y + (Y/4 - Y/100 + Y/400))/365.2425 + ([Y/4] - [Y/100] + [Y/400] - (Y/4 - Y/100 + Y/400) + D)/365.2425)
=INT((365.2425*Y)/365.2425 + ([Y/4] - Y/4 - [Y/100] + Y/100 + [Y/400] - Y/400 + D)/365.2425)
=INT(Y + ([Y/4] - Y/4 - [Y/100] + Y/100 + [Y/400] - Y/400 + D)/365.2425)
=Y + INT(([Y/4] - Y/4 - [Y/100] + Y/100 + [Y/400] - Y/400 + D)/365.2425)

The difference between Y and this approximation is our error function and is equal to

INT(([Y/4] - Y/4 - [Y/100] + Y/100 + [Y/400] - Y/400 + D)/365.2425)

Note that the difference function has the same value if a multiple of 400 is added to Y; i.e. only the values of Y modulo 400 with values between 0 and 399 need be examined when the error function is evaluated.

Unfortunately, the error function has a minimum value of -0.001301274553891091 at Y modulo 400= 303 (a leap year) and D = 1. The error function has a maximum value of 1.001973218175465 at Y modulo 400 = 95 (a leap year) and D = 366. Since the difference between maximum and minimum values is greater than 1, a constant cannot be subtracted to make our error function go to zero.

The values that are out of range could be special cased or corrections applied once the number of days contained in the years from the base date are subtracted. Instead, the approach taken here is to determine either the number of full centuries represented by the time span or the number of full 400 year periods. This allows the values [Y/100] and [Y/400] to be eliminated in the difference function and this creates a function that is better behaved.

3.1.2.2 Average days per century approximation

The number of full centuries from the base date to the given date will now be determined.

Let Y be the year (for years starting March 1) of our given date. Then [Y/100] is the number of full centuries and the value to be determined given the number of days between base and target date. Since the number of full century years is to be determined given the days since our base date and there are 365.2425*100 such days per century, consider the difference between [Y/100] and

INT(INT((365*Y + [Y/4] - [Y/100] + [Y/400] + D)/365.2425)/100)
=INT((365*Y + [Y/4] - [Y/100] + [Y/400] + D)/(100*365.2425))
In particular, the difference function equal to

-[Y/100] + ((365*Y + [Y/4] - [Y/100] + [Y/400] + D)/(100*365.2425))

is examined.

First, notice that this difference function has the same value if a multiple of 400 is added to Y; i.e., only the values of Y modulo 400 with values between 0 and 399 need be examined. This difference function is our error function and it has a minimum value of 0.000006904926745444993 when Y mod 400 = 300 (not a leap year), D=1. The error function has a maximum value of 1.000000080212121 when Y mod 400 = 399 (a leap year), D=366. In order for our approximation to be valid, the error function must have values that are greater than or equal to 0 and less than 1 so that applying the greatest integer function to it produces a result of zero. This can be done by subtracting a constant K:

((365*Y + [Y/4] - [Y/100] + [Y/400] + D - K)/(100*365.2425))

This requires

K/(100*365.2425) > 1.000000080212121 - 1.0

so that the error function doesn't exceed 1. Solving for K,

K > 0.002929687499688476

The value of the error function can be decreased as long as it is not less than zero. Thus

K/(100*365.2425) $\leq$ 0.000006904926745444993

and solving for K

K $\leq$ 0.2521972656249999

Thus
0.002929687499688476 < K $\leq$ 0 .2521972656249999

Therefore,

[Y/100] = INT((365*Y + [Y/4] - [Y/100] + [Y/400] + D - K)/(100*365.2425))

where

0.002929687499688476 < K $\leq$ 0.2521972656249999

3.1.2.3 Days per 400 years approximation

One could similarly evaluate [Y/400] using

=INT((365*Y + [Y/4] - [Y/100] + [Y/400] + D - K)/(400*365.2425))

where

$0.002929687515908501 < K \leq 1$

However, once [Y/100] has been determined, [[Y/100]/4] = [Y/400] can be easily calculated.

3.1.2.4 *Application of century and 400 year approximations to determine the year*

Since an appropriate constant K can be chosen to make our century and 400 year approximations exactly correct, the day count of

365*Y + [Y/4] -[Y/100] + [Y/400] + D

can be converted into

365*Y +[Y/4] + D

by adding our value for [Y/100] and subtracting our value for [Y/400]. Our task is now to find Y given

365*Y +[Y/4] + D

This can be done using the approximation

[(365*Y +[Y/4] + D)/365.25]
=[(365*Y + 0.25*Y - 0.25*Y+[Y/4] + D)/365.25]
=[(365.25*Y - 0.25*Y+[Y/4] + D)/365.25]
=Y + [([Y/4] - Y/4 + D)/365.25]

Examining the error function

([Y/4] - Y/4 + D)/365.25

reveals that it has a minimum of 0.000684462696783017 that occurs both at leap year 259 day 1 and non-leap year 299 day 1. The maximum is 1 at leap year 3 day 366. Thus the error function's value is reduced to zero when the greatest integer function is applied to that value if a constant K is first subtracted:

([Y/4] - Y/4 + D - K)/365.25

where

$0 < K \leq 365.25*0.000684462696783017 = 0.25$

3.1.2.5 Alternative application of the century to determine the year

Once the number of full centuries have been determined, this number can be used to convert the day count to an interval less than a century and this interval will not require accounting for the troublesome century leap years. In other words subtracting

365.2425*100 *[Y/100]

from the day count. If variable "I" represents this number of days within a century (assuming a year starting March 1 of the year 0 and the full 100 year intervals were subtracted relative to that date), then

INT(I/365.25)= [I/365.25]

gives the year number except when I is a multiple of 4. When I is a multiple of 4, then the interval represents a time period that includes, as its last day, a leap day. In this case I is exactly divisible by 365.25 (that is, the remainder is zero) but the year value is one less than the quotient. It can be shown that the function:

[I/(365.25 + 1/4000)]= [I/365.25]

provided I is not a multiple of 4 and is less than 365.25*100 and
if I is a multiple of 4 then

[I/365.25] = [I/(365.25 + 1/4000)] + 1

The function [I/(365.25 + 1/4000)] therefore satisfies the requirement.

3.1.3 Determining the Day of the Year

If Z is the number of days since our base year and Y the year of the date to be determined, then in the Gregorian Calendar, there will be 365.25*Y - [Y/100] + [Y/400] days up to January 1 of year Y and

D = Z - INT(365.25*Y - [Y/100] + [Y/400])

where D is the day number within Y of our Gregorian target date.

Notice that if in the process of calculating a Gregorian date,

365*Y +[Y/4] + D

has been calculated then the year day number D can be calculated by subtracting 365.25*Y.

3.1.4 Determining the Month

The number of days in the months prior to the month of our date are now accounted for given the day number since March 1 of the date's year. Writing out the range and domain of the function explicitly:

days in previous months	**month**
0	3
31	4
61	5
92	6
122	7
153	8
184	9
214	10

245	11
275	12
306	13
337	14

Since, all the months except February are either 30 or 31 days long, one might suspect that a linear approximation can be used and the fractional parts removed to generate a convenient form of the function. One cannot be certain that this will work on a particular function before the situation is examined in more detail, but, as will be shown, the function given in the table can be successfully treated this way.

The function will take the form:
FIX(M*X+B)
where
X = day number,
M = the slope of the line,
B= the Y intercept of the line, and
FIX() is the truncation function.

Now since FIX() removes the fraction, the values of MX+B that have the same resulting function, are values from some integer, say Y, up to, but not including, the value Y+1. The range of values for each X can be graphed, but because of the scale, it would be difficult to see precisely how the required line would fit through all the ranges for each function value or whether or not it is even possible to do so. However, it isn't difficult to write a computer program that determines what equations can pass through these intervals. Doing so reveals that the line with maximum slope goes from point (123,7) to (337,14), has slope 0.03271028037383177 and Y intercept = 2.976635514018692. The line with minimum slope goes from (276,12) to (31,4), has a slope = .0326530612244898 and Y intercept = 2.987755102040816. These two lines cross at (194.3333333333375, 9.333333333333469) and I call this the pivot point, since there will be lines that pass within the specified intervals, go through this point, and take on any of the slopes from .0326530612244898 to 0.03271028037383177. Given slope M, the value

9.333333333333469 - M * 194.3333333333375

can be used as the Y intercept. However, there will usually be many other Y intercept values that work as well. For example, a slope of 0.03268167079916079 has a range of Y intercepts from 2.980154491703223 to 2.986276940682814.

This information can be used to look for a function in the form:

(AX + B)\C

where A, B, and C are integers. In particular, small values of C and values of C that are powers of 2 were searched for so that the division could be effected by shifts. Some of the many possible values are displayed in the following table:

C	A	B range (value of upper bound can not be used)
153	5	455.9999999999999 to 456.9999999999999
306	10	911.9999999999999 to 913.9999999999998
367	12	1093 to 1094
398	13	1188 to 1189
16384	535	48948 to 48951
32768	1070	97896 to 97902
32768	1071	97643 to 97825
65536	2140	195792 to 195804
65536	2141	195516 to 195773
65536	2142	195286 to 195650
65536	2143	195163 to 195313

3.1.5 Determining the Day

Once the month number has been identified, the remaining days can be calculated by subtracting the number of days in the prior months. There are many ways to calculate the function which converts month number to days in prior months. This function was examined as part of algorithm development for converting from a Gregorian or Julian Date to the corresponding Julian Day number and will not be reproduced here.

3.2 Julian Day Number to Julian Date - Algorithm Development

3.2.1 Choosing a Base Date

Algorithm development begins by choosing a base date that our day count will be relative to instead of the less convenient Julian Day Number 0. Since conversion will be to a Julian date, this base date will be the Julian year zero.

The Julian Day Number is converted to the number of days from a base date of March 1 in the year zero. This creates a calendar with January and February as months 13 and 14 rather than months 1 and 2. The purpose of this scheme is to cause any extra day in a leap year to occur at the end of the "year." As will be seen, this simplifies the calculation of the number of leap years from year zero to a particular date and it simplifies the function which relates the number of days in the preceding months to the month number.

Let MOD be the modulo function. Julian year Y is a leap year if Y MOD 4 = 0. In our year starting on March 1, the days in that year span two different years that start on January 1. For example, if February 29 is in year Y for a year beginning January 1, it is in year Y-1 for a year beginning March 1. Similarly, to see if there is an extra day due to a leap year in year Y of a calendar beginning March 1, then our Julian leap year test must be applied to year Y+1.

From Table 2, it can be seen that in order to convert our Julian Day Number to the number of days since March 1 of the year zero in the Julian calendar, 1721117.5 must be subtracted. If day 1 is to be **on** March 1 of the year zero in the Julian, 1721116.5 would be subtracted.

3.2.2 Determining the Year

With the number of days now relative to our base of March 1 of the year 0, the number of full years within the given time span in days is determined. Given year Y, the number of days to account for is 365*Y+[Y/4] = [365.25*Y] where [x] is the greatest integer function. Here the extra leap year days are added because the date for a calendar beginning March 1 places February 29 in the preceding year.

If Z is the number of days since our base year and Y the year of the date to be determined, then

D = Z - 365.25*Y

will be the day number within year Y of our Julian Target date. The number of days accounted for by a span of Y years, although equal to 365*Y+[Y/4], is approximately equal to 365*Y+Y/4 = 365.24*Y since the greatest integer function changes the value of its argument by less than 1. If D is the day number of year Y with values 1 through 365 (or 366 if a leap year), then the task is to find Y given

365*Y+[Y/4] + D days.

Y must then be approximately

INT((365*Y+[Y/4] + D)/365.25)

Examining the error from this approximation:

INT((365*Y+[Y/4] + D)/365.25) = INT((365*Y+[Y/4] + D)/365.25)
=[(365*Y + 0.25*Y - 0.25*Y+[Y/4] + D)/365.25]
=[(365.25*Y - 0.25*Y+[Y/4] + D)/365.25]
=Y + [([Y/4] - Y/4 + D)/365.25]

Examining the error function

([Y/4] - Y/4 + D)/365.25

reveals that it has a minimum of 0.000684462696783017 that occurs both at leap year 259 day 1 and non-leap year 299 day 1. The maximum is 1 at leap year 3 day 366. Thus the error function's value is reduced to zero when the greatest integer function is applied to that value if a constant K is first subtracted:

([Y/4] - Y/4 + D - K)/365.25

where

$0 < K \leq 365.25*0.000684462696783017 = 0.25$

3.2.3 Determining the Month and Day

Once the year has been determined, the procedure for finding the month and day is exactly the same as that described in the previous section for converting a Julian Day Number to a Gregorian date.

3.3 Julian Day Number to Julian and Gregorian Date - Algorithm Forms

3.3.1 Julian Day number to Gregorian Date

The previous algorithm development section justifies the following general algorithm forms for converting a Julian Day Number into a Gregorian Date. In this description:

[x] = the greatest integer that does not exceed x. For example, [-1.5]=-2. This is sometimes called the floor function (for example in C/C++).

INT(x) = [x] NOTE: some computer languages have a different definition.

FIX(x) = the number x without its fraction. For example, FIX(-1.5)=-1

x\y = FIX(x/y)

STEP 1 Let JD be the Julian Day Number. Calculate

Z = INT(JD - 1721118.5)

R = the fractional part of JD - 1721118.5 = (JD - 1721118.5) - Z

STEP 2 Calculate the value of A which is the number of full centuries:

A = INT(Z - K1)/(36524.25))

where 0.002929687499688476 < K1 ≤ 0.2521972656249999

STEP 3 Calculate the days within the whole centuries (in the Julian Calendar) by adding back days removed in the Gregorian Calendar. The value of B is this number of days minus a constant.

B = Z - K2+ A - INT(A/4)

where 0 < K2 ≤ 0.25

(It is sometimes convenient to make K1 = K2 = 0.25.)

STEP 4 Calculate the value of Y, the year in a calendar whose years start on March 1:

Y = INT(B/365.25)

STEP 5 Calculate the value of the day count in the current year using **ONE** of the following:

1. C = Z + A - INT(A/4) - INT(365.25*Y)
2. C = FIX(B - INT(365.25*Y)) + 1
   This form especially convenient in languages where the assignment of a real to an integer rounds up because B and therefore (B - INT(365.25*Y)) always have a fractional part of 0.75 or greater. In such languages,

   C = B - INT(365.25*Y)
   where C is an integer and B is a real.

STEP 6 Calculate the value of the month in the current year using **ONE** of the following:

1. M = FIX((5*C + 456)/ 153)
2. M = FIX((535*C + 48950)/ 16384)

   (Note that 16384 is 2^14 and the division can be implemented with shifts if the number has a binary representation.)

3. M = FIX((2140*C + 195800)/ 65536)

   (Note that 65536 is 2^16 and the can be implemented by taking the upper 16 bits of a 32 bit number.)

4. M = FIX(K3*C+9.333333333333469 - K3 * 194.3333333333375)

   where K3 = .0326530612244898 to 0.03271028037383177

STEP 7 Calculate the value of F, which is the number of days in the preceding months in the current year, using **ONE** of the following methods:

1. F is the value of a vector indexed by M. This vector has values 0, 31, 61, 92, 122, 153, 184, 214, 245, 275, 306, 337. (M has values that range from 3 through 14.) This method is usually the fastest method and often takes less computer memory to implement.
2. F = (153*M-457)\5
3. F = (979*M-2918)\32
   This allows a shift to carry out the division.
4. If real arithmetic is desired,
   F=FIX(30.6*M-91.4) = -122+FIX(30.6*(M+1))
   can be used although real arithmetic is usually slower and takes more code space. Other coefficients are possible as described in the previous section. Note that since M is 3 or greater, 30.6*M-91.4 is always positive. Either FIX or the greatest integer function can be used.

STEP 8 The Gregorian date's day of the month is

Day = C - F + R

STEP 9 Convert the month and year to a calendar starting January 1, using **ONE** of the following methods:

1. If M>12 then
   Y = Y + 1
   M = M - 12
   end if
2. Y=Y + FIX(M/13)
   M = M - 12*FIX(M/13)

The Gregorian date has
Year = Y
Month = M

This calculation is valid for any Julian Day Number including negative JDN and produces a Gregorian date (or possibly a proleptic Gregorian date).

Example

The following sequence calculates Gregorian year Y, month M, and day D from a Julian Day Number:

Z = INT(JD - 1721118.5)	step 1
R = JD - 1721118.5 - Z	step 1
G = Z - .25	used in later steps
A = INT(G / 36524.25)	step 2
B = A - INT(A / 4)	part of step 3
year = INT((B+G) / 365.25)	part of step 3 and step 4
C = B + Z - INT(365.25 * year)	step 5
month = FIX((5 * C + 456) / 153)	step 6
day = C - FIX((153 * month - 457) / 5) + R	step 7 and 8
IF month > 12 THEN	step 9
year = year + 1	step 9
month = month - 12	step 9
END IF	step 9

3.3.2 Julian Day number to Julian Date

The previous algorithm development section justifies the following general algorithm forms for converting a Julian Day Number into a Julian Date. In this description:

[x] = the greatest integer that does not exceed x. For example, [-1.5]=-2. This is sometimes called the floor function (for example in C/C++).

INT(x) = [x] NOTE: some computer languages have a different definition.

FIX(x) = the number x without its fraction. For example, [-1.5]=-1

x\y = FIX(x/y)

STEP 1 Let JD be the Julian Day Number. Calculate

Z = INT(JD - 1721116.5)

R = the fractional part of JD - 1721116.5 = (JD - 1721116.5) - Z

STEP 2 Calculate the value of Y, the year in a calendar whose years start on March 1:

Y = INT((Z - K)/365.25)
where $0 < K \leq 0.25$

STEP 3 Calculate the value of the day count in the current year:

C = Z - INT(365.25*Y)

STEP 4 Calculate the value of the month in the current year using **ONE** of the following:

1. M = FIX((5*C + 456)/ 153)
2. M = FIX((535*C + 48950)/ 16384)

   (Note that 16384 is 2^14 and the division can be implemented with shifts if the number has a binary representation.)

3. M = FIX((2140*C + 195800)/ 65536)

   (Note that 65536 is 2^16 and the can be implemented by taking the upper 16 bits of a 32 bit number.)

4. M = FIX(K3*C+9.333333333333469 - K3 * 194.3333333333375)

   where K3 = .0326530612244898 to 0.03271028037383177

STEP 5 Calculate the value of F, which is the number of days in the preceding months in the current year, using **ONE** of the following methods:

1. F is the value of a vector indexed by M. This vector has values 0, 31, 61, 92, 122, 153, 184, 214, 245, 275, 306, 337. (M has values that range from 3 through 14.) This method is usually the fastest method and often takes less computer memory to implement.
2. F = (153*M-457)\5
3. F = (979*M-2918)\32
   This allows a shift to carry out the division.
4. If real arithmetic is desired,
   F=FIX(30.6*M-91.4) = -122+FIX(30.6*(M+1))
   can be used although real arithmetic is usually slower and takes more code space. Other coefficients are possible as described in the previous section. Note that since M is 3 or greater, 30.6*M-91.4 is always positive. Either FIX or the greatest integer function can be used.

STEP 6 The Julian date's day of the month is

Day = C - F + R

STEP 7 Convert the month and year to a calendar starting January 1, using **ONE** of the following methods:

1. If M>12 then
   Y = Y + 1
   M = M - 12
   end if
2. Y=Y + FIX(M/13)
   M = M - 12*FIX(M/13)

The Julian date has
Year = Y
Month = M

This calculation is valid for any Julian Day Number including negative JDN and produces a Julian date (or possibly a proleptic Julian date).

Example

The following sequence calculates Julian year Y, month M, and day D from a Julian Day Number:

Z = INT(JD - 1721116.5)	step 1
R = JD - 1721116.5 - Z	step 1
year = INT((Z -0.25) / 365.25)	step 2
C = Z - INT(365.25 * year)	step 3
month = FIX((5 * C + 456) / 153)	step 4
day = C - FIX((153 * month - 457) / 5) + R	step 5 and 6
IF month > 12 THEN	step 7
year = year + 1	step 7
month = month - 12	step 7
END IF	step 7

3.4 Julian Day Number to Gregorian and Julian Dates - Implementation

3.4.1 Choices and Notation

3.4.1.1 Choices to be made

The following decisions have to be made regarding any implementation:

- Are dates specified in the Gregorian or Julian Calendar?
- What range of dates are to be supported?
- What number word size will be used?
- Is the greatest integer function available or is only the truncation function?
- Is real arithmetic available and required?
- Is conditional execution such as "IF/THEN" supported?

3.4.1.2 Notation

[x] = the greatest integer that does not exceed x. For example, [-1.5]=-2. This is sometimes called the floor function (for example in C/C++).

INT(x) = [x] NOTE: some computer languages have a different definition.

FIX(x) = the number x without its fraction. For example, FIX(-1.5)=-1

x\y = FIX(x/y)

3.4.2 Julian Day Number to Gregorian Date Implementation Examples

1. The following sequence calculates Gregorian year Y, month M, and day D from Julian Day Number JD. It uses real arithmetic.

```
Z = INT(JD - 1721118.5)
R = JD - 1721118.5 - Z
G = Z - .25
A = INT(G / 36524.25)
B = A - INT(A / 4)
year = INT((B+G) / 365.25)
C = B + Z - INT(365.25 * year)
month = FIX((5 * C + 456) / 153)
day = C - FIX((153 * month - 457) / 5) + R
IF month > 12 THEN
    year = year + 1
    month = month - 12
END IF
```

COMMENTS:

- The range of years that are supported is only limited by the word size of the implementation.
- Although FIX((5*C + 456)/ 153) and FIX((153*M-457)/5) are elegant expressions, the use of indexed arrays are usually a faster implementation.
- Some languages only implement, within the language proper, the greatest integer function for a real argument. One can, of course, always create this function as part of the program itself.

2. The following sequence calculates Gregorian year Y, month M, and day D from Julian Day Number JD. It uses integer arithmetic except for the values of JD, R, and Day since the Julian Day Number may be fractional.

```
Z = INT(JD - 1721118.5)
R = JD - 1721118.5 - Z
H = 100*Z - 25
A = INT(H / 3652425)
B = A - INT(A / 4)
year = INT((100*B+H) / 36525)
C = B + Z - 365 * year - INT( year/4)
month = FIX((5 * C + 456) / 153)
day = C - FIX((153 * month - 457) / 5) + R
IF month > 12 THEN
    year = year + 1
    month = month - 12
END IF
```

COMMENTS:

- The range of years that are supported is only limited by the word size of the implementation.
- The greatest integer function can be replaced by integer division if JD ≥ March 1 of the year zero.
- Although FIX((5*C + 456)/ 153) and FIX((153*M-457)/5) are elegant expressions, the use of indexed arrays are usually a faster implementation.
- Some languages only implement the greatest integer function for a real argument.

3. The following sequence calculates Gregorian year Y, month M, and day D from Julian Day Number JD. It uses integer arithmetic except for the values of JD, R, and Day since the Julian Day Number may be fractional. It uses integer division and the shift right operator to effect integer division by powers of 2.

   ****WARNING*** Only valid for dates greater than or equal to 1721118.25 (April 28.75 of the year zero).

```
Z = INT(JD - 1721118.5)
R = JD - 1721118.5 - Z
H = 100*Z - 25
A = H \ 3652425
B = A - SHIFTRIGHT(A,2)
year = (100*B+H) \ 36525
C = B + Z - 365 * year - SHIFTRIGHT( year,2)
month = SHIFTRIGHT((535*C + 48950),14)
day = C - SHIFTRIGHT( (979*M-2918),5) + R
IF month > 12 THEN
    year = year + 1
    month = month - 12
END IF
```

   COMMENTS:

   - Only valid for dates greater than or equal to 1721118.25 (April 28.75 of the year zero).
   - Although SHIFTRIGHT((535*C + 48950),14) and SHIFTRIGHT((979*M-2918),5) are elegant expressions, the use of indexed arrays are usually a faster implementation.

4. QBASIC Implementation as a subroutine - using real arithmetic:

   Note: & indicates 32-bit (long) integer and # indicates double precision real

```
SUB jdtog1 (jd AS DOUBLE, year AS LONG, month AS INTEGER, day AS DOUBLE)
'INPUT: jd = Julian Day Number
'OUTPUT: Gregorian year, month, and day
Z& = INT(jd - 1721118.5#) 'Day 1 is on March 1 of the year 0
R# = jd - 1721118.5# - Z& 'the fractional part of the day
G# = Z& - 0.25#
A& = INT(G# / 36524.25#) 'whole centuries
B& = A& - INT(A& / 4#) 'Julian days in whole centuries - 0.25
```

```
year = INT((G#+B&) / 365.25#)
C& = Z& +B& - INT(365.25 * year) 'days in the year
month = (5 * C& + 456) \ 153
day = C& - ((153 * month - 457) \ 5) + R#
IF month > 12 THEN 'convert to year that starts January 1
    year = year + 1
    month = month - 12
END IF 'convert to year that starts January 1
END SUB
```

5. QBASIC Implementation as a subroutine - Integer arithmetic version:

```
SUB jdtog2 (JD AS DOUBLE, year AS LONG, month AS INTEGER, day AS DOUBLE)
'***WARNING*** ONLY VALID IF JD ≥ 1721118.25 (February 28.75 of year zero)
'INPUT: jd = Julian Day Number
'OUTPUT: Gregorian year, month, and day
Z& = INT(JD - 1721118.5) 'Day 1 is on March 1 of the year 0
R# = JD - 1721118.5 - Z& 'the fractional part of the day
H& = 100*Z& - 25
A& = H&\3652425 'whole centuries
B& = A& - A&\4 'adjustment to find the Julian days in whole centuries
year = (100*B&+H&) \ 36525
C& = B& + Z& - 365 * year - year\4 'days in the year
month = (5 * C& + 456) \ 153
day = C& - (153 * month - 457) \ 5 + R#
IF month > 12 THEN 'convert to year that starts January 1
    year = year + 1
    month = month - 12
END IF 'convert to year that starts January 1
END SUB
```

6. In *Astronomical Algorithms*, 1991, page 63, Jean Meeus (essentially) gives the following algorithm. This can be shown to be an instance of our derived algorithm provided the Julian Day numbers is greater than or equal to 1867215.75, that is, for dates from February 29.25, 400 in the proleptic Gregorian calendar.

```
Z = FIX(JD +0.5)
R = JD + 0.5 - Z
G = FIX((Z - 1867216.25)/36524.25)
A = Z + 1 + G - FIX(G/4)
B = A + 1524
C = FIX((B - 122.1)/365.25)
D = FIX(365.25*C)
E = FIX((B - D)/30.6001)
Day of month = B - D - FIX(30.6001*E) + R
If E
```

```
    Month = E - 1
    Year = C - 4716
else
    Month = E - 13
    Year = C - 4715
end if
```

COMMENTS:

- This algorithm is only valid for Julian Day numbers greater than or equal to 1867216.25, that is, dates from February 29.75, 400 in the proleptic Gregorian calendar. Note that Meeus states that his algorithm is valid for all non-negative Julian Day numbers but, in fact, his algorithm switches to another form that finds the Julian Date rather than the Gregorian date if the Julian Day Number represents a date before the Gregorian Reform.
- If FIX is replaced with the greatest integer function, the algorithm is valid for all Julian Day Numbers within the range of the word size.
- Meeus states that the 30.6001 in the algorithm can't be replaced by 30.6. An algorithm could be developed that uses a slope of 1/30.6 = 0.03267973856209151 by using a linear function with a Y intercept within the range of 2.980392156862745 to 2.986928104575163.

7. For dates in the Gregorian calendar at noon, *The Explanatory Supplement to the Astronomical Almanac*,1992, page 604, gives an algorithm by Fliegel and Van Flandern that calculates dates after November 23, -4713, given an integral Julian Day Number. This algorithm can be shown to be an instance of the general form described in the algorithm development section.

```
L = JD + 68569
N = FIX((4*L)/146097)
L = L - FIX((146097 * N + 3)/4)
I = FIX(4000 *(L + 1)/1461001)
L = L - FIX((1461*I)/4) + 31
J = FIX((80*L)/2447)
D = L - FIX((2447*J)/80)
L=FIX(J/11)
M = J + 2 - 12*L
Y = 100*(N-49) + I + L
```

COMMENTS:

- The algorithm is not valid for dates before November 23, -4713 (negative JD numbers).
- The algorithm could be easily modified to handle Julian Day Numbers with a non-zero fractional part. The other algorithms in this section show how this can be done.
- The algorithm is designed so that it can be directly implemented using integer arithmetic and the usual integer division.
- This algorithm uses the alternative method of determining the year given the number of whole centuries as was described in section 3.1.2.5.

8. Python Code Example for JDN to Gregorian
WARNING Note that
    - the Python // implements the greatest integer function, instead of the INT() used in this document and
    - the Python int() implements the FIX() notation used in this document.

```
def JDNtoGregorian(JDN):
    """Julian Day Number to Gregorian Date"""
    T=JDN - 1721118.5
    Z = T//1
    R = T - Z
    G = Z - .25
    A = G // 36524.25
    B = A - A // 4
    year = int((B+G) // 365.25)
    C = B + Z - (365.25 * year)//1
    month = int((5 * C + 456) / 153)
    day = C - int((153 * month - 457) // 5) + R
    if month > 12:
        year+=1
        month-=12
    return([month,day,year])
```

3.4.3 Julian Day Number to Julian date - Implementation Examples

1. The following sequence calculates Julian year Y, month M, and day D from Julian Day Number JD. It uses real arithmetic.

```
Z = INT(JD - 1721116.5)
R = JD - 1721116.5 - Z
year = INT((Z - 0.25) / 365.25)
C = Z - INT(365.25 * year)
month = FIX((5 * C + 456) / 153)
day = C - FIX((153 * month - 457) / 5) + R
IF month > 12 THEN
      year = year + 1
      month = month - 12
END IF
```

COMMENTS:

- The range of years that are supported is only limited by the word size of the implementation.
- Although FIX((5*C + 456)/ 153) and FIX((153*M - 457)/5) are elegant expressions, the use of indexed arrays are usually a faster implementation.
- Some languages only implement, within the language proper, the greatest integer function for a real argument. One can, of course, always create this function as part of the program itself.

2. The following sequence calculates Julian year Y, month M, and day D from Julian Day Number JD. It uses integer arithmetic except for the values of JD, R, and Day since the Julian Day Number may be fractional.

```
Z = INT(JD - 1721116.5)
R = JD - 1721116.5 - Z
year = INT((100*Z - 25) / 36525)
C = Z - 365 * year - INT( year/4)
month = FIX((5 * C + 456) / 153)
day = C - FIX((153 * month - 457) / 5) + R
IF month > 12 THEN
    year = year + 1
    month = month - 12
END IF
```

COMMENTS:

- The range of years that are supported is only limited by the word size of the implementation.
- The greatest integer function can be replaced by integer division if JD ≥ April 29 of the year zero.
- Although FIX((5*C + 456)/ 153) and FIX((153*M - 457)/5) are elegant expressions, the use of indexed arrays are usually a faster implementation.
- Some languages only implement the greatest integer function for a real argument.

3. The following sequence calculates Julian year Y, month M, and day D from Julian Day Number JD. It uses integer arithmetic except for the values of JD, R, and Day since the Julian Day Number may be fractional. It uses integer division and the shift right operator to effect integer division by powers of 2.

****WARNING*** Only valid for dates greater than or equal to 1721116.25 (February 28.75 of the year zero).

```
Z = INT(JD - 1721116.5)
R = JD - 1721116.5 - Z
year = FIX((100*Z - 25)) / 36525)
C = Z - 365 * year - SHIFTRIGHT( year,2)
month = SHIFTRIGHT((535*C + 48950),14)
day = C - SHIFTRIGHT( (979*M-2918),5) + R
IF month > 12 THEN
    year = year + 1
    month = month - 12
END IF
```

COMMENTS:

- Only valid for dates greater than or equal to 1721116.25 (February 28.75 of the year zero).
- Although SHIFTRIGHT((535*C + 48950),14) and SHIFTRIGHT((979*M-2918),5) are elegant expressions, the use of indexed arrays are usually a faster implementation.

4. QBASIC Implementation as a subroutine - using real arithmetic:

   Note: & indicates 32-bit (long) integer and # indicates double precision real

```
SUB jdtoj1 (jd AS DOUBLE, year AS LONG, month AS INTEGER, day AS DOUBLE)
'INPUT: jd = Julian Day Number
'OUTPUT: Julian year, month, and day
Z& = INT(jd - 1721116.5#) 'Day 1 is on March 1 of the year 0
R# = jd - 1721116.5# - Z& 'the fractional part of the day
year = INT((Z& - 0.25#) / 365.25#)
C& = Z& - INT(365.25 * year) 'days in the year
month = (5 * C& + 456) \ 153
day = C& - ((153 * month - 457) \ 5) + R#
IF month > 12 THEN 'convert to year that starts January 1
      year = year + 1
      month = month - 12
END IF 'convert to year that starts January 1
END SUB
```

5. QBASIC Implementation as a subroutine - Integer arithmetic version:

```
SUB jdtoj2 (JD AS DOUBLE, year AS LONG, month AS INTEGER, day AS DOUBLE)
'***WARNING*** ONLY VALID IF JD ≥ 1721116.5 (April 29 of day zero or greater)
'INPUT: jd = Julian Day Number
'OUTPUT: Julian year, month, and day
Z& = INT(JD - 1721116.5) 'Day 1 is on March 1 of the year 0
R# = JD - 1721116.5 - Z& 'the fractional part of the day
year = (100*Z& - 25) \ 36525
C& = Z& - 365 * year - year\4 'days in the year
month = (5 * C& + 456) \ 153
day = C& - (153 * month - 457) \ 5 + R#
IF month > 12 THEN 'convert to year that starts January 1
      year = year + 1
      month = month - 12
END IF 'convert to year that starts January 1
END SUB
```

6. In *Astronomical Algorithms*, 1991, page 63, Jean Meeus (essentially) gives the following algorithm. This can be shown to be an instance of the previously derived algorithm provided the Julian Day numbers is greater than or equal to 1867215.75, that is, for dates from February 29.25, 400 in the proleptic Gregorian calendar.

```
Z = FIX(JD +0.5)
R = JD + 0.5 - Z
```

```
B = Z + 1524
C = FIX((B - 122.1)/365.25)
D = FIX(365.25*C)
E = FIX((B - D)/30.6001)
Day of month = B - D - FIX(30.6001*E) + R
If E
    Month = E - 1
    Year = C - 4716
else
    Month = E - 13
    Year = C - 4715
end if
```

COMMENTS:

- This algorithm is only valid for Julian Day numbers greater than or equal to -0.5, that is, dates from noon of January 1, -4712 in the proleptic Julian calendar. Note that Meeus states that his algorithm is valid for all non-negative Julian Day numbers but that his algorithm switches to another form that finds the Julian Date rather than the Gregorian date if the Julian Day Number represents a date before the Gregorian Reform.
- If FIX is replaced with the greatest integer function, the algorithm is valid for all Julian Day Numbers within the range of the word size.
- Meeus states that the 30.6001 in the algorithm can't be replaced by 30.6. An algorithm could be developed that uses a slope of 1/30.6 = 0.03267973856209151 by using a linear function with a Y intercept within the range of 2.980392156862745 to 2.986928104575163.

3.4.4 Introduction to Analysis of Date Algorithms

3.4.4.1 Analysis of Julian Day Number to Gregorian date algorithms

This section analyzes the core of various conversion algorithms that convert from the Julian Day Number to a Gregorian Date. Since all of the algorithms that deal with fractional Julian Day Numbers initially remove the fractional part at the beginning of the procedure and account for it at the end of the procedure, this part of the algorithm is ignored. Also excluded is the conversion from a calendar beginning March 1 to a calendar beginning January 1 because the choices available are well known and removal means that the choice between an "if-test" and an arithmetic operation can be ignored. The analysis is further simplified by ignoring the fact that the functions that convert between month number and the number of days in the preceding month can be implemented as an indexed array.

The intent of the following table is to give some idea of the complexity of some of the various implementations that have been presented. The ID numbers reference algorithms in the implementation section. The table shows that the algorithms are roughly comparable in complexity. This means that in order to find the "best" algorithm for the implementation on a particular machine, it is necessary to look more closely at the speed of various operations and the operation of the algorithm a whole. The following section looks at some timing for various implementations using QBASIC on a PC under Microsoft Windows.

Analysis of Julian Day Number to Gregorian Date Algorithms

ID	AUTHOR	TYPE	RANGE	+ -	*	\	/	SHIFT	INT	FIX
1	Baum	Real	All	8	3	2	3	0	4	2
2	Baum	Integer	All	9	5	6	0	0	4	2
3	Baum	Integer	≥1721118.25	9	5	2	0	4	0	2
6	Meeus	Real	≥1867215.75	11	2	1	3	0	0	6
7	Fliegel	Integer	≥0.5	9	7	6	0	0	0	6

3.4.4.2 QBASIC Timing of some Julian Day Number to Gregorian date implementations

Routine	Type	Range	Conversions per second	Comments
Baum4	Integer	>1721119	3542.3	No fractional JD allowed
FLIEGEL	Integer	≥0.5	3194.0	No fractional JD allowed
BAUM3	Integer	≥ 1721118.5	2392.3	
BAUM1	Real	All	1202.6	Tricky Roundup
BAUM2	Real	All	1183.0	
Meeus	Real	≥1867215.75	1010.9	

```
SUB baum1 (jd AS DOUBLE, year AS LONG, month AS INTEGER, day AS DOUBLE)
'Baum - real arithmetic with tricky roundup
Z& = INT(jd - 1721118.5#)
R# = jd - 1721118.5# - Z&
G# = Z& - .25#
A& = INT(G# / 36524.25#)
B# = G# + A& - INT(A& / 4#)'days if years were Julian
year = INT((B#) / 365.25#)
C& = B# - INT(365.25 * year) 'B# ends in 0.75 and integer conversion rounds up
month = (5 * C& + 456) \ 153
day = C& - ((153 * month - 457) \ 5) + R#
IF month > 12 THEN
year = year + 1
month = month - 12
END IF
END SUB

SUB baum2 (jd AS DOUBLE, year AS LONG, month AS INTEGER, day AS DOUBLE)
'INPUT: jd = Julian Day Number
'OUTPUT: Gregorian year, month, and day
'standard real arithmetic
Z& = INT(jd - 1721118.5#) 'Day 1 is on March 1 of the year 0
R# = jd - 1721118.5# - Z& 'the fractional part of the day
G# = Z& - .25#
A& = INT(G# / 36524.25#) 'whole centuries
B& = A& - INT(A& / 4#) 'Julian days in whole centuries - 0.25
year = INT((G# + B&) / 365.25#)
C& = Z& + B& - INT(365.25 * year)'days in the year
month = (5 * C& + 456) \ 153
day = C& - ((153 * month - 457) \ 5) + R#
IF month > 12 THEN 'convert to year that starts January 1
year = year + 1
month = month - 12
END IF 'convert to year that starts January 1
END SUB
```

```
SUB baum3 (jd AS DOUBLE, year AS LONG, month AS INTEGER, day AS DOUBLE)
'***WARNING*** ONLY VALID IF JD ≥ 1721118.5 (March 1 of day zero)
'INPUT: jd = Julian Day Number
'OUTPUT: Gregorian year, month, and day
'integer arithmetic and integer division in particular
Z& = INT(jd - 1721118.5#) 'Day 1 is on March 1 of the year 0
R# = jd - 1721118.5# - Z& 'the fractional part of the day
H& = 100 * Z& - 25
A& = H& \ 3652425'whole centuries
B& = A& - A& \ 4'adjustment to find the Julian days in whole centuries
year = (100 * B& + H&) \ 36525
C& = B& + Z& - 365 * year - year \ 4'days in the year
month = (5 * C& + 456) \ 153
day = C& - (153 * month - 457) \ 5 + R#
IF month > 12 THEN 'convert to year that starts January 1
year = year + 1
month = month - 12
END IF 'convert to year that starts January 1
END SUB

SUB baum4 (jd AS DOUBLE, year AS LONG, month AS INTEGER, day AS DOUBLE)
'NO FRACTIONAL JULIAN DAY NUMBER ALLOWED
'JULIAN DAY NUMBER MUST BE GREATER THAN 1721119
Z& = jd - 1721119
H& = 100 * Z& - 25
A& = H& \ 3652425
B& = A& - A& \ 4
year = (100 * B& + H&) \ 36525
C& = B& + Z& - 365 * year - year \ 4
month = (5 * C& + 456) \ 153
day = C& - (153 * month - 457) \ 5
IF month > 12 THEN
year = year + 1
month = month - 12
END IF 'convert to year that starts January 1
END SUB

SUB fliegel (jd AS DOUBLE, year AS LONG, month AS INTEGER, day AS DOUBLE)
l& = jd + 68569
n& = (4 * l&) \ 146097
l& = l& - (146097 * n& + 3) \ 4
I& = 4000 * (l& + 1) \ 1461001
l& = l& - (1461 * I&) \ 4 + 31
J& = (80 * l&) \ 2447
day = l& - (2447 * J&) \ 80
l& = J& \ 11
month = J& + 2 - 12 * l&
year = 100 * (n& - 49) + I& + l&
END SUB
```

```
SUB meeus (jd AS DOUBLE, year AS LONG, month AS INTEGER, day AS DOUBLE)
'meeus
Z& = INT(jd + .5#)
R# = jd + .5# - Z&
G& = INT((Z& - 1867216.25#) / 36524.25#)
A& = Z& + 1 + G& - INT(G& / 4)
B& = A& + 1524
C& = INT((B& - 122.1#) / 365.25#)
d& = INT(365.25# * C&)
E& = INT((B& - d&) / 30.6001#)
day = B& - d& - INT(30.6001# * E&) + R#
IF E& < 14 THEN
month = E& - 1
ELSE 'E=14 or E=15
month = E& - 13
END IF
IF month > 2 THEN
year = C& - 4716
ELSE
year = C& - 4715 ' M=1 or M=2
END IF
END SUB
```

4. Modified Julian Day

The **Modified Julian Day** or **MJD** is defined as:

MJD= JD - 2400000.5

so that MJD = 0.0 corresponds to November 17.0, 1858. The MJD begins at Greenwich mean midnight and is sometimes used in reference to artificial satellites.

5. Rata Die

5.1 Rata Die Algorithms

The Rata Die is a count of the number of days since a base date of December 31 of the year zero in the proleptic Gregorian calendar. Thus Rata Die day one occurs on January 1 of the year 1 which begins at Julian Day Number 1721425.5. The algorithm for calculating the Rata Die given a year, month, and day in the Gregorian Calendar is the same as calculating the Julian Day Number except that in the final step, the number of days since March 1 of the year 0 is converted to the number of days since December 31 of the year zero rather than the number of days since November 24.5 in the year -4713. Since there are 306 days between March 1 and December 31, the conversion consists of subtracting 306 from the day count relative to March 1 of the year 0. That is, March 1 of the year zero is Rata Die day -306.

The following is a general procedure for calculating the Rata Die:

STEP 1 Calculate the value of Z using **ONE** of the following methods:

1. Z = Y + (M-14)\12
2. IF M<3 THEN Z=Y-1 ELSE Z=Y
3. On some computers, we can place the month and year in memory locations so that a subtraction of the month will carry into the year when the month is January or February.

STEP 2 Calculate the value of F using **ONE** of the following methods:

1. F is the value of a vector indexed by M. This vector has values 306, 337, 0, 31, 61, 92, 122, 153, 184, 214, 245, 275. This method is usually the fastest method and often takes less computer memory to implement.
2. IF M<3 THEN M=M+12
   F = (153*M-457)\5
3. IF M<3 THEN M=M+12
   F = (979*M-2918)\32
   This allows a shift to carry out the division.
4. F = (979*(M-12*((M-14)\12))-2918)\32
   This eliminates the need for an IF test and carries out the division using a shift.

STEP 3 The Rata Die is
D+F+365*Z+Z\4-Z\100+Z\400 - 306

Therefore, a typical algorithm will comprise:

```
IF m < 3 THEN
      m = m + 12
      y = y - 1
END IF
rd = d + (153 * m - 457) \ 5 + 365 * y + y \ 4 - y \ 100 + y \ 400 - 306
```

where
d= day of the month
m = month
y = year
(in the Gregorian calendar) and
rd is the Rata Die.

5.2 Rata Die Implementation Examples

1. The following is an implementation as a subroutine in QBASIC. It is valid for all dates up to the limits of the QBASIC implementation for long integers (32 bit signed arithmetic).

```
SUB rata (rd AS LONG, year AS LONG, month AS INTEGER, day AS INTEGER)
'INPUT: year, month, day which represents a Gregorian Date
'OUTPUT: Rata Die number
y& = year 'Do not modify input parameter
m% = month 'Do not modify input parameter
IF m% < 3 THEN
    m% = m% + 12
    y& = y& - 1
END IF
rd = day + (153 * m% - 457) \ 5 + INT(365.25# * y&) - INT(y& *.01) + INT(y& *.0025) - 306
END SUB
```

2. The following is an implementation as a subroutine in QBASIC. It is only valid for non-negative Rata Die.

```
SUB rata (rd AS LONG, year AS LONG, month AS INTEGER, day AS INTEGER)
'INPUT: year, month, day which represents a Gregorian Date
'OUTPUT: Rata Die number
'***WARNING*** only valid for non-negative Rata Die,
' i.e., dates from January 1 of the year 1 or later
y& = year 'Do not modify input parameter
m% = month 'Do not modify input parameter
IF m% < 3 THEN
    m% = m% + 12
    y& = y& - 1
END IF
rd = day + (153 * m% - 457) \ 5 + 365& * y& + y& \ 4 - y& \ 100 + y& \ 400 - 306
END SUB
```

6. Inverse Rata Die

6.1 Inverse Rata Die Algorithms

The algorithm development section for converting a Julian Day Number into a Gregorian date justifies most of the following general algorithm forms for converting a Rata Die number into a Gregorian Date. The only modification is in the first step where, since Rata Die day 1 is January 1 of the year 1, the constant 306 is added to convert to the number of days since March 1 of the year 0.

In this description:

> **[x]** = the greatest integer that does not exceed x. For example, [-1.5]=-2. This is sometimes called the floor function (for example in C/C++).
>
> **INT(x)** = [x] NOTE: some computer languages have a different definition.
>
> **FIX(x)** = the number x without its fraction. For example, FIX(-1.5)=-1
>
> **x\y** = FIX(x/y)

STEP 1 Let RD be the Rata Day Number. Calculate

> Z = RD + 306

STEP 2 Calculate the value of A which is the number of full centuries:

> A = INT(Z - K1)/(36524.25))
>
> where 0.002929687499688476 < K1 $\leq$ 0.2521972656249999

STEP 3 Calculate the days within the whole centuries (in the Julian Calendar) by adding back days removed in the Gregorian Calendar. The value of B is this number of days minus a constant.

> B = Z - K2+ A - INT(A/4) =
>
> where 0 < K2 $\leq$ 0.25
>
> (It is sometimes convenient to make K1 = K2 = 0.25.)

STEP 4 Calculate the value of Y, the year in a calendar whose years start on March 1:

> Y = INT(B/365.25)

STEP 5 Calculate the value of the day count in the current year using **ONE** of the following:

1. C = Z + A - INT(A/4) - INT(365.25*Y)
2. C = FIX(B - INT(365.25*Y)) + 1
   This form especially convenient in languages where the assignment of a real to an integer rounds up because B and therefore (B - INT(365.25*Y)) always have a fractional part of 0.75 or greater. In such languages,

   C = B - INT(365.25*Y)
   where C is an integer and B is a real.

STEP 6 Calculate the value of the month in the current year using **ONE** of the following:

1. M = FIX((5*C + 456)/ 153)
2. M = FIX((535*C + 48950)/ 16384)

   (Note that 16384 is 2^14 and the division can be implemented with shifts if the number has a binary representation.)

3. M = FIX((2140*C + 195800)/ 65536)

   (Note that 65536 is 2^16 and the can be implemented by taking the upper 16 bits of a 32 bit number.)

4. M = FIX(K3*C+9.333333333333469 - K3 * 194.3333333333375)

   where K3 = .0326530612244898 to 0.03271028037383177

STEP 7 Calculate the value of F, which is the number of days in the preceding months in the current year, using **ONE** of the following methods:

1. F is the value of a vector indexed by M. This vector has values 0, 31, 61, 92, 122, 153, 184, 214, 245, 275, 306, 337. (M has values that range from 3 through 14.) This method is usually the fastest method and often takes less computer memory to implement.
2. F = (153*M-457)\5
3. F = (979*M-2918)\32
   This allows a shift to carry out the division.
4. If real arithmetic is desired,
   F=FIX(30.6*M-91.4) = -122+FIX(30.6*(M+1))
   can be used although real arithmetic is usually slower and takes more code space. Other coefficients are possible as described in the previous section. Note that since M is 3 or greater, 30.6*M-91.4 is always positive. Either FIX or the greatest integer function can be used.

STEP 8 The Gregorian date's day of the month is

Day = C - F

STEP 9 Convert the month and year to a calendar starting January 1, using **ONE** of the following methods:

1. If M>12 then
   Y = Y + 1

```
        M = M - 12
    end if
2.  Y=Y + FIX(M/13)
    M = M - 12*FIX(M/13)
```

The Gregorian date has
Year = Y
Month = M

This calculation is valid for any Rata Die number including negative Rata Die numbers and produces a Gregorian date (or possibly a proleptic Gregorian date).

Example

The following sequence calculates Gregorian year Y, month M, and day D from a Rata Die number:

Z = RD + 306	step 1
G = Z - .25	used in later steps
A = INT(G / 36524.25)	step 2
B = A - INT(A / 4)	part of step 3
year = INT((B+G) / 365.25)	part of step 3 and step 4
C = B + Z - INT(365.25 * year)	step 5
month = FIX((5 * C + 456) / 153)	step 6
day = C - FIX((153 * month - 457) / 5)	step 7 and 8
IF month > 12 THEN	step 9
year = year + 1	step 9
month = month - 12	step 9
END IF	step 9

6.2 Inverse Rata Die Implementation Examples

6.2.0 Notation

[x] = the greatest integer that does not exceed x. For example, [-1.5]=-2. This is sometimes called the floor function (for example in C/C++).

INT(x) = [x] NOTE: some computer languages have a different definition.

FIX(x) = the number x without its fraction. For example, FIX(-1.5)=-1

x\y = FIX(x/y)

6.2.1 Rata Die Number to Gregorian Date

1. The following sequence calculates Gregorian year Y, month M, and day D from Rata Die number RD. It uses real arithmetic.

   ```
   Z = RD + 306
   G = Z - .25
   A = INT(G / 36524.25)
   B = A - INT(A / 4)
   year = INT((B+G) / 365.25)
   C = B + Z - INT(365.25 * year)
   month = FIX((5 * C + 456) / 153)
   day = C - FIX((153 * month - 457) / 5)
   IF month > 12 THEN
       year = year + 1
       month = month - 12
   END IF
   ```

   COMMENTS:

   - The range of years that are supported is only limited by the word size of the implementation.
   - Although FIX((5*C + 456)/ 153) and FIX((153*M - 457)/5) are elegant expressions, the use of indexed arrays are usually a faster implementation.
   - Some languages only implement, within the language proper, the greatest integer function for a real argument. One can, of course, always create this function as part of the program itself.

2. The following sequence calculates Gregorian year Y, month M, and day D from Rata Die number RD. It uses integer arithmetic.

   ```
   Z = RD + 306
   H = 100*Z - 25
   A = INT(H / 3652425)
   B = A - INT(A / 4)
   year = INT((100*B+H) / 36525)
   ```

```
C = B + Z - 365 * year - INT( year/4)
month = FIX((5 * C + 456) / 153)
day = C - FIX((153 * month - 457) / 5)
IF month > 12 THEN
    year = year + 1
    month = month - 12
END IF
```

COMMENTS:

- The range of years that are supported is only limited by the word size of the implementation.
- The greatest integer function can be replaced by integer division if RD > March 1 of the year zero.
- Although FIX((5*C + 456)/ 153) and FIX((153*M-457)/5) are elegant expressions, the use of indexed arrays are usually a faster implementation.
- Some languages only implement the greatest integer function for a real argument.

3. The following sequence calculates Gregorian year Y, month M, and day D from Rata Die number RD. It uses integer division and the shift right operator to effect integer division by powers of 2.

****WARNING*** Only valid for RD greater than or equal to -305 (March 2 of the year zero).

```
Z = RD + 306
H = 100*Z - 25
A = H \ 3652425)
B = A - SHIFTRIGHT(A,2)
year = (100*B+H)) \ 36525
C = B + Z - 365 * year - SHIFTRIGHT( year,2)
month = SHIFTRIGHT((535*C + 48950),14)
day = C - SHIFTRIGHT( (979*M-2918),5)
IF month > 12 THEN
    year = year + 1
    month = month - 12
END IF
```

COMMENTS:

- Only valid for RD greater than -306 (March 1 of the year zero).
- Although SHIFTRIGHT((535*C + 48950),14) and SHIFTRIGHT((979*M-2918),5) are elegant expressions, the use of indexed arrays are usually a faster implementation.

4. QBASIC Implementation as a subroutine - using real arithmetic:

   Note: & indicates 32-bit (long) integer and # indicates double precision real

```
SUB inrata (rd AS LONG, year AS LONG, month AS INTEGER, day AS INTEGER)
'INPUT: rd = Rata Die Number
'OUTPUT: Gregorian year, month, and day
'standard real arithmetic
Z& = rd + 306 'Day 1 is on January 1 of the year 1
G# = Z& - .25#
A& = INT(G# / 36524.25#) 'whole centuries
B& = A& - INT(A& / 4#) 'Julian days in whole centuries - 0.25
year = INT((G# + B&) / 365.25#)
C& = Z& + B& - INT(365.25 * year)'days in the year
month = (5 * C& + 456) \ 153
day = C& - ((153 * month - 457) \ 5)
IF month > 12 THEN 'convert to year that starts January 1
    year = year + 1
    month = month - 12
END IF 'convert to year that starts January 1
END SUB
```

5. QBASIC Implementation as a subroutine - Integer arithmetic version:

```
SUB inrata2 (rd AS LONG, year AS LONG, month AS INTEGER, day AS INTEGER) 'INPUT: rd =
Rata Die Number 'OUTPUT: Gregorian year, month, and day
'***WARNING*** ONLY VALID IF RD > -306 (March 1 of year zero)
'INPUT: rd = Julian Day Number
'OUTPUT: Gregorian year, month, and day
Z& = rd + 306 'Day 1 is on March 1 of the year 0
H& = 100*Z& - 25
A& = H&\3652425 'whole centuries
B& = A& - A&\4 'adjustment to find the Julian days in whole centuries
year = (100*B&+H&) \ 36525
C& = B& + Z& - 365 * year - year\4 'days in the year
month = (5 * C& + 456) \ 153
day = C& - (153 * month - 457) \ 5
IF month > 12 THEN 'convert to year that starts January 1
     year = year + 1
     month = month - 12
END IF 'convert to year that starts January 1
END SUB
```

7. Day of the Week

7.1 Day of the Week Algorithms

In this section, the convention is used that the numbers from 0 through 6 correspond to the day of the week names as follows:

0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

Given the beginning of a day (0.00 hours) whose date is expressed as Julian Day Number JD (fractional part 0.5), the day of the week is equal to

(INT(JD + 1.5)) modulo 7

It is written in this form because some software implementations of the modulo function round up any fractional argument. Note also that some modulo implementations do not assure that the result is positive when the argument is negative. Microsoft's Visual Basic is an example of an implementation that suffers from both problems.

If the fractional part of the Julian Day number for a day beginning at 0 hours is truncated (removed or made zero) it represents noon of the previous day. If a date expressed as such a Julian Day number is used, then the day of the week is equal to: (INT(JD + 2)) modulo 7.

For example,

Friday, December 31, 1999 at noon is JD 2451544
Saturday, January 1, 2000 0 hours is JD 2451544.5
Saturday, January 1, 2000 at noon is JD 2451545

INT(2451544+2) = INT(2451544.5 + 1.5) = 2451546

and 2451546 mod 7 = (350220*7+6) mod 7 = 6 (Saturday)

7.2 Day of Week Implementations

The following algorithms give the day of the week number (Sunday = 0) given a Gregorian Date.

1. The following QBASIC implementation as a subroutine is valid for all Gregorian Dates up to the word limits of the implementation.

```
SUB week1 (dow AS INTEGER, y1 AS LONG, m1 AS INTEGER, d1 AS INTEGER)
m% = m1
y& = y1
IF m% < 3 THEN
    y& = y& - 1
    m% = m% + 12
END IF
dow = (d1 + (153 * m% - 457) \ 5 + INT(365.25# * y&) - INT(y& * .01) + INT(y& * .0025) + 2) MOD
7
IF dow<0 THEN
    dow=dow+7
END IF
END SUB
```

2. The following QBASIC implementation as a subroutine is valid for all Gregorian Dates from March 1 of the year 0 or greater. It uses integer arithmetic

```
SUB week2 (dow AS INTEGER, y1 AS LONG, m1 AS INTEGER, d1 AS INTEGER)
m% = m1
y& = y1
IF m% < 3 THEN
    y& = y& - 1
    m% = m% + 12
END IF
dow = (d1 + (153 * m% - 457) \ 5 + 365 * y& +y& \ 4 - y& \ 100 + y& \ 400 + 2) MOD 7
END SUB
```

3. A typical Visual Basic subroutine

```
Sub JDtoWeek(jd As Double)
'INPUT: jd = Julian Day Number
'OUTPUT: modifies the caption of label dow
i% = Int(jd + 1.5) Mod 7
If i% < 0 Then i% = i% + 7
dow.Caption = weekname(i%)
End Sub
```

4. Typical Python code

```
weekdayNames=["Sunday","Monday","Tuesday", "Wednesday","Thursday","Friday","Saturday"]
weekday=int((JDN+2)%7)
print("weekday = ",weekdayNames[weekday])
```

8. Difference Between Dates

Calculating the difference between two dates is easy once these dates are expressed as Julian Day Numbers - simply take the difference between the Julian Day Numbers for each date. One can use the algorithms given in previous sections to perform this conversion. Care must be taken to use the algorithm appropriate to the calendar of each of the dates - either Gregorian or Julian. If both dates are in the same calendar, then the constants cancel and further algorithm simplifications are possible.

9. (year, ordinal) to JDN

The year and ordinal to Julian Day Number is straightforward. The code follows almost exactly the Gregorian to JDN routine except that we make the month and day equal to 1 and this allows us to simplify. In particular, since with $M = 1 < 3$ the routine then makes

$M = M + 12$ and

$Y = Y - 1$

and with M having value 13,

$(153 * M - 457)\backslash 5 = (153 * 13 - 457)\backslash 5 = 306$

We therefore change the constant

1721118.5

to

$1721118.5 + 306 = 1721424.5$.

Here is the resulting code in Python. Note that

- the Python // implements the greatest integer function, instead of the INT() used in this document and
- the Python int() implements the FIX() notation used in this document.

```
def YearOrdtoJDN(year,ordinal):
    """(year,ordinal) to Julian Day Number"""
    year-=1
    return(ordinal + 365 * year + year // 4 - year // 100 + year // 400 + 1721424.5)
```

10. JDN to (year, ordinal)

The Julian Day Number to (year, ordinal) algorithm shares a great deal with the JDN to Gregorian algorithms, with the code up to the year calculation the same. The day count is then adjusted to start on January 1 and that adjustment depends upon whether or not the year is a leap year. Here is a Python routine that implements this algorithm.

Note that

- the Python // implements the greatest integer function, instead of the INT() used in this document,
- the Python int() implements the FIX() notation used in this document, and
- the Python % implements the modulo function.

```
def JDNtoYearOrd(JDN):
    """Julian Day Number to year and ordinal"""
    T=JDN - 1721118.5
    Z = T//1
    R = T - Z
    G = Z - .25
    A = G // 36524.25
    B = A - A // 4
    year=int((B+G) // 365.25)
    C = B + Z - (365.25 * year)//1+R
    if year%4: # not a leap year; most common case
        C+=59
        if C>365:
            C-=365
            year+=1
    elif year%100: # is a leap year; second most common case
        C+=60
        if C>366:
            C-=366
            year+=1
    elif year%400: # is not a leap year; 3rd most common case
        C+=59
        if C>365:
            C-=365
            year+=1
    else: # is a leap year: least common case
        C+=60
        if C>366:
            C-=366
            year+=1
    return (year,C)
```

11. Additional Proofs

11.1 Proof of Meeus' Julian Day Number to Gregorian Date algorithm

In *Astronomical Algorithms*, 1991, page 63, Jean Meeus gives the algorithm (essentially) as

Z = FIX(JD +0.5)
R = JD + 0.5 - Z
G = FIX((Z - 1867216.25)/36524.25)
A = Z + 1 + G - FIX(G/4)
B = A + 1524
C = FIX((B - 122.1)/365.25)
D = FIX(365.25*C)
E = FIX((B - D)/30.6001)
Day of month = B - D - FIX(30.6001*E) + R
If E
    Month = E - 1
    Year = C - 4716
else
    Month = E - 13
    Year = C - 4715
end if

The following shows that this is an instance of the derived algorithm provided the Julian Day numbers is greater than or equal to 1867215.75, that is, for dates from February 29.25, 400 in the proleptic Gregorian calendar.

Proof that the year is correct

Z = FIX(JD + 0.5) = FIX(JD + 0.5) - 1 + 1= FIX(JD - 0.5) + 1
Z = FIX(JD - 0.5 - 1721118.0) + 1721118.0 + 1
Z = FIX(JD - 1721118.5) + 1721119.0

It was given that
G = FIX((Z- 1867216.25)/36524.25)

Since INT(JD - 1721118.5) is the Z in the derived algorithm, we write

G = FIX((Z + 1721119.0 - 1867216.25)/36524.25)
where Z is from the derived algorithm.
G = FIX((Z + 1721119.0 - 1867216.25)/36524.25)
G = FIX((Z -146097.25)/36524.25)
G = FIX((Z -146097.25)/36524.25)
= FIX((Z - 4*36524.25 - 0.25)/36524.25)
= - 4 + FIX((Z - 0.25)/36524.25)

Expressed using the Z in the derived algorithm, it was given that

A = Z + 1721119.0 + 1 + G - FIX(G/4)

and combined with

B = A + 1524,

we have
B = Z + 1721119.0 + 1 + G - FIX(G/4) + 1524,

which when combined with G becomes

B = Z + 1722641+ FIX((Z - 0.25)/36524.25) - FIX(FIX((Z - 0.25)/36524.25)/4)
and since

C = FIX((B - 122.1)/365.25)
C = FIX((Z + 1722641 - 122.1+ FIX((Z - 0.25)/36524.25) - FIX(FIX((Z - 0.25)/36524.25)/4))/365.25)
C = FIX((Z + 4716*365.25 - 0.1+ FIX((Z - 0.25)/36524.25) - FIX(FIX((Z - 0.25)/36524.25)/4))/365.25)
C = 4716 + FIX((Z - 0.1+ FIX((Z - 0.25)/36524.25) - FIX(FIX((Z - 0.25)/36524.25)/4))/365.25)

All but 4716 of C is step 2 of the derived algorithm with K1 = 0.25 and K2 = 0.1 The 4716 is removed as part of the year adjustment that depends upon the month number.

Proof that the month is correct

The following shows that the month calculation is equivalent to the derived algorithm. It was given that

E = FIX((B - D)/30.6001)

The previous section gives B in terms of the "Z" used in the derived algorithm:

B = Z + 1722641+ FIX((Z - 0.25)/36524.25) - FIX(FIX((Z - 0.25)/36524.25)/4)

and using
D = FIX(365.25*C)
substitution gives
E = FIX((Z + 1722641+ FIX((Z - 0.25)/36524.25) - FIX(FIX((Z - 0.25)/36524.25)/4) -
FIX(365.25*C))/30.6001)

Since C is the year plus 4716, D is a day count plus 4716*365.25 = 1722519. Subtracting this out from the expression for E and using C' to represent the year gives
E' = FIX((Z + 122+ FIX((Z - 0.25)/36524.25) - FIX(FIX((Z - 0.25)/36524.25)/4) - FIX(365.25*C'))/30.6001)

Noting that
(Z +FIX((Z - 0.25)/36524.25) - FIX(FIX((Z - 0.25)/36524.25)/4) - FIX(365.25*C')
is the day count, the function is essentially

FIX((count +122)/30.6001)

This is the value for the month number + 1, because
month +1 = FIX((count +122)/30.6001)
month = FIX((count +91.39999)/30.6001)
month = FIX((count +91.39999)/30.6001)
month = FIX((count*0.032679727 + 2.986918017))

And this is a specific instance of our derived formula for the month. In fact, given the slope, the constant can have any value between 2.980395348 and 2.986928463.

The fact that E is the month +1 value rather than the month is adjusted in the section that converts the date to a calendar that starts on January 1.

Proof that the day is correct

It was previously shown that FIX(30.6001*E) is the number of days in the preceding months. Since B - D is the number of days in the year,

Day of month = B - D - FIX(30.6001*E) + R

11.2 Proof of Explanatory Supplement Gregorian to Julian Day Number algorithm

For dates in the Gregorian calendar at noon, *The Explanatory Supplement to the Astronomical Almanac*,1992, page 604, gives the value for the Gregorian Date (after November 23, -4713) given a Julian Day Number as:

(Only valid for JD $\geq$ and JD assumed to be at noon of that day)
L = JD + 68569
N = (4*L)/146097
L = L - (146097 * N + 3)/4
I = (4000 *(L + 1)/1461001
L = L - (1461*I)/4 + 31
J = (80*L)/2447
D = L - (2447*J)/80
L=J/11
M = J + 2 - 12*L
Y = 100*(N-49) + I + L

Converting the assigned variables to distinct names so that they may be more easily referenced:

L1 = JD + 68569
N = (4*L1)/146097
L2 = L1 - (146097 * N + 3)/4
I = (4000 *(L2 + 1)/1461001
L 3= L2 - (1461*I)/4 + 31

J = (80*L3)/2447
D = L3 - (2447*J)/80
L4=J/11
M = J + 2 - 12*L4
Y = 100*(N-49) + I + L4

Using the value of L1, N can be written as:

N = (4*(JD + 68569))/146097
N = (4*JD +4* 68569)/146097
N = (4*JD + 274276)/146097
N = (4*JD + 274276)/(365.2425*400)
N = (JD + 68569)/(365.2425*100)
N = (JD - 1721118.5 + 1721118.5 + 68569)/(365.2425*100)
N = (JD - 1721118.5 + 1789687.5)/(36524.25)
N = (JD - 1721118.5 + 1789688.25 - 0.75)/36524.25
N = (JD - 1721118.5 + 49*36524.25 - 0.75)/36524.25

But JD in the formula represents a day (for this algorithm) that spans two Gregorian days. The JD number represents noon of the second day, the first day of which starts at INT(JD) - 0.5 since no time is specified. Converting to a Julian Day Number that represents the start of the day gives
N = (JD - 0.5 + 0.5 - 1721118.5 + 49*36524.25 - 0.75)/(36524.25)
N = (JD - 0.5 - 1721118.5 + 49*36524.25 - 0.25)/(36524.25)
N = 49 + (JD - 0.5 - 1721118.5 - 0.25)/(36524.25)
so N-49 = # of 100 year periods

L2 = L1 - (146097 * N + 3)/4
L2 = L1 - (365.2425*400 * N + 3)/4
L2 = L1 - 365.2425*100 * N - 0.75

since L1 = JD + 68569 = (JD - 0.5 - 1721118.5 + 49*36524.25 - 0.25)

L2 = L1 - 365.2425*100 * N - 0.75
L2 = (JD - 0.5 - 1721118.5 + 49*36524.25 - 0.25) - 365.2425*100 * N - 0.75
L2 = (JD - 0.5 - 1721118.5 + 49*36524.25 - 0.25) - 365.2425*100 * N - 0.75
Using the value for N
L2 = (JD - 0.5 - 1721118.5 - 1.0) - 365.2425*100 * ((JD - 0.5 - 1721118.5 - 0.25)/(36524.25))
so L2 is 1 less than the number of days left when the number of days in the whole 100 year periods is subtracted.

I = (4000 *(L2 + 1)/1461001
I = (4000 *(L2 + 1)/(365.25*4000 + 1)
I = ((L2 + 1)/(365.25 + 1/4000)
I = number of years in the 100 year remainder, as previously shown in the algorithm development section.

L 3= L2 - (1461*I)/4 + 31
L 3= L2 - (4*365.25*I)/4 + 31
L 3= L2 - 365.25*I + 31

J = (80*L3)/2447 where J = month - 2
J = (80*(L2 - 365.25*I + 31))/2447
J = (80*(L2 - 365.25*I) + 80*31)/2447
J = (80*(L2 - 365.25*I) + 2480)/2447
J = (80*(L2 - 365.25*I) + 2480)/2447
J =0.03269309358398*(L2 - 365.25*I) + 1.0134859011033

Since L2 is one less than the number of days left after subtracting the days in full 100 year periods, and 365.25*I the number of days in the remaining full years,

(L2 - 365.25*I) = day of year - 1
J =0.03269309358398*(day of year -1) + 1.0134859011033
J =0.03269309358398*(day of year) + 1.0134859011033 - 0.03269309358398
J =0.03269309358398*(day of year) + 0.98079280751932

and since J = month - 2

month - 2 = 0.03269309358398*(day of year) + 0.98079280751932
month = 0.03269309358398*(day of year) + 2.98079280751932

This is simply the previously derived function that converts day of year to month number. Here the slope is 0.03269309358398 and the Y intercept can range anywhere from 2.97874948917046 to 2.982427462198739.

D = L3 - (2447*J)/80
D = L3 - (2447*(month-2))/80
D = L3 - ((2447*month-2447*2))/80
D = L3 - ((2447*month - 4894))/80
D = L3 - FIX(30.5875*month - 61.175)
D = (L2 - 365.25*I + 31) - FIX(30.5875*month - 61.175)

Since L2 is one less than the number of days left after subtracting the days in full 100 year periods, and 365.25*I the number of days in the remaining full years,

(L2 - 365.25*I + 31) = day of year - 1 + 31 = day of year + 30.
therefore

D = day of year + 30 - FIX(30.5875*month - 61.175)
D = day of year - FIX(30.5875*month - 91.175)

But, as was shown in the development section,
FIX(30.5875*month - 91.175)
is the function that gives the number of days in the preceding months. In fact, if the slope = 30.5875 the Y intercept can be anywhere in the range of -91.22499465942383 to -91.11249732971191

L4=J/11
J= month - 2 so L4 = 1 if month 13 or 14

Finally, converting to a year starting January 1.
J+2 = month
and subtracting 12 if that number is 13 or 14.
Therefore

M = J + 2 - 12*L4

N-49 = # of 100 year periods
I = year modulo 100
and adding one if the month is 13 or 14

Y = 100*(N-49) + I + L4

11.3 Proof regarding [I/365.25]

The following shows that [I/365.25] = [I/(365.25 + 1/4000)]
provided I is not a multiple of 4 and is less than 365.25*100 and
if I is a multiple of 4 then
[I/365.25] = [I/(365.25 + 1/4000)]+ 1

Let
I = 365.25*K + R
where K is an integer and R a real and
$0 \leq R < 365.25$

Since I and K are integers, R must also be a multiple of 0.25

[I/365.25] = [((365.25*K + R)/365.25] = K

[I/(365.25 + 1/4000)]
=[(365.25*K + R)/(365.25 + 1/4000)]
=[(365.25*K + R)/365.25025]
=[365.25/365.25025*K + R/365.25025]
=[0.9999993155377717058 3*K + R/365.25025]
=[(+1 - 1 + 0.9999993155377717058 3)*K + R/365.25025]
=[(+1 - 0.000000684462228294 17)*K + R/365.25025]
=[K - 0.000000684462228294 17*K + R/365.25025]
=K + [- 0.000000684462228294 17*K + R/365.25025]
=K + [-K/1461001 + R/365.25025]

Now
[-K/1461001 + R/365.25025] = -1
provided
R = 0 and
$1 \leq K \leq 1461001$

and
[-K/1461001 + R/365.25025] = 0
provided
1 ≤ R < 365.25
under certain conditions of K. Let's look at the case where R = 0.25:

0 ≤ -K/1461001 + 0.25/365.25025 < 1
0 ≤ -K/1461001 +0.000684462228294162700 7 < 1
- 0.000684462228294162700 7 ≤ -K/1461001 < 1- 0.000684462228294162700 7
- 0.000684462228294162700 7 ≤ -K/1461001 < 0.999315537771705837299 3
-100 ≤ -K <1460001
- 1460001 < K ≤100

Now if R > 0.25 and K is positive, it can be even larger than 100. Therefore

[I/365.25] = [I/(365.25 + 1/4000)]
provided I is not a multiple of 4 and is less than 365.25*100 and
if I is a multiple of 4 then
[I/365.25] = [I/(365.25 + 1/4000)]+ 1

11.4 Proof of 3 useful identities

The following identities are proven below:

1. (1461*(Y+4800))\4 = FIX(365.25*Y) + 1753200
2. 367*(M-2))\12 = FIX(30.583333333*M - 91.166666666) + 30
3. -(3*((Y+4900)\100))\4 = -FIX(Y/100) + FIX(Y/4) - 36

Proof of 1.

(1461*(Y+4800))\4
=FIX((1461*(Y+4800))/4)
=FIX(((1461/4)*(Y+4800)))
=FIX((365.25*(Y+4800)))
=FIX(365.25*Y+365.25*4800)
=FIX(365.25*Y+1753200)
=FIX(365.25*Y)+1753200

Proof of 2.

Since 367*(-2)/12=-61.166666666
and 367/12 = 30.583333333

367*(M-2))\12
= FIX(30.583333333*M - 61.166666666)
= 30 + FIX(30.583333333*M - 91.166666666)

Proof of 3.

-(3*((Y+4900)\100))\4 = -(3*(FIX(Y/100+49)))\4
= -(3*(49+FIX(Y/100)))\4
= -((147+3*FIX(Y/100)))\4
= -(FIX(147/4+3*FIX(Y/100)/4))
= -(FIX(36.75+3*FIX(Y/100)/4))
= -(36+FIX(0.75+(3/4)*FIX(Y/100)))
= -36 -FIX(0.75+(1-(1/4))*FIX(Y/100))
= -36 -FIX(0.75 -(1/4)*FIX(Y/100)+FIX(Y/100))
= -36 -FIX(0.75 -(1/4)*FIX(Y/100)+FIX(Y/100))
= -36 -FIX(0.75 -FIX(Y/400)+FIX(Y/100))
= -36 -FIX(0.75) +FIX(Y/400) - FIX(Y/100)
= -FIX(Y/100) + FIX(Y/4) - 36

12. Revision List

- Version 1.003 to 2
  Tuesday, September 3, 2019
  Section 7.2; item 1; 4th line from bottom. Added missing text "<0 THEN"
  Comment: this must have been some kind of conversion error, because the original .htm file was correct
- Version 2 to 3
  Tuesday, March 3, 2020
  Seven instances of "IF M<3 THEN Z=Y-1 ELSE Z=Y" or "IF M<3 THEN M=M+12" were incorrectly changed from the original .htm to "IF M"
- Version 3 to 4 Sunday, October 18, 2020
  - Added Python code for JDN to Gregorian and Gregorian to JDN
  - Added Python code (year, ordinal) to JDN and JDN to (year, ordinal) Thanks to Jacob Pratt for suggesting that code involving (year, ordinal) be considered.
  - Added Python code for weekday calculations
  - page 15
    changed section 3.1.1.2.1 Average days per year approximation

    From: The number of days accounted for by a span of Y years, although equal to 365*Y+[Y/4] -[Y/100]+[Y/400], is approximately equal to 365*Y+Y/4 -Y/100+Y/400 = 365.2425 since the greatest integer function changes the value of its argument by less than 1.

    To: The number of days accounted for by a span of Y years, although equal to 365*Y+[Y/4] -[Y/100]+[Y/400], is approximately equal to 365*Y+Y/4 -Y/100+Y/400 = Y*(365 + 1/4 - 1/100 +1/400) = Y*365.2425 since the greatest integer function changes the value of its argument by less than 1.
  - Page 16 third line
    From: examing
    To: examining
  - Minor typos fixed
  - Added true table of contents
- Version 4 to 5
  - Corrected (year,ordinal) to JDN Python routine and showed how it is derived from a Gregorian to JDN algorithm. Thanks to Jacob Pratt bringing the error to my attention.
  - Fixed implementation headings
  - Added pointer on first page to this revision list section

[END OF DOCUMENT]