MMX - time of the seasons

```
' seasons.bas         March 6, 2017
' calendar date and UTC time of the seasons
' Micromite eXtreme version
''''''''''''''''''''''''''''
' dimension global arrays
dim sl(50) as float, sr(50) as float, sa(50) as float, sb(50) as float
dim jdleap(28) as float, leapsec(28) as float, month$(12) as string
' global constants
const pi2 = 2.0 * pi, pidiv2 = 0.5 * pi, dtr = pi / 180.0
' read solar ephemeris data
for i% = 1 to 50
  read sl(i%), sr(i%), sa(i%), sb(i%)
next i%
data 403406,      0, 4.721964,      1.621043
data 195207, -97597, 5.937458,  62830.348067
data 119433, -59715, 1.115589,  62830.821524
data 112392, -56188, 5.781616,  62829.634302
data   3891,  -1556, 5.5474  , 125660.5691
data   2819,  -1126, 1.5120  , 125660.9845
data   1721,   -861, 4.1897  ,  62832.4766
data      0,    941, 1.163   ,      0.813
data    660,   -264, 5.415   , 125659.310
data    350,   -163, 4.315   ,  57533.850
data    334,      0, 4.553   ,    -33.931
data    314,    309, 5.198   , 777137.715
data    268,   -158, 5.989   ,  78604.191
data    242,      0, 2.911   ,      5.412
data    234,    -54, 1.423   ,  39302.098
data    158,      0, 0.061   ,    -34.861
data    132,    -93, 2.317   , 115067.698
data    129,    -20, 3.193   ,  15774.337
data    114,      0, 2.828   ,   5296.670
data     99,    -47, 0.52    ,  58849.27
data     93,      0, 4.65    ,   5296.11
data     86,      0, 4.35    ,  -3980.70
data     78,    -33, 2.75    ,  52237.69
data     72,    -32, 4.50    ,  55076.47
data     68,      0, 3.23    ,    261.08
data     64,    -10, 1.22    ,  15773.85
data     46,    -16, 0.14    , 188491.03
data     38,      0, 3.44    ,  -7756.55
data     37,      0, 4.37    ,    264.89
data     32,    -24, 1.14    , 117906.27
data     29,    -13, 2.84    ,  55075.75
data     28,      0, 5.96    ,  -7961.39
data     27,     -9, 5.09    , 188489.81
data     27,      0, 1.72    ,   2132.19
data     25,    -17, 2.56    , 109771.03
```

```
  data     24,    -11, 1.92    ,  54868.56
  data     21,      0, 0.09    ,  25443.93
  data     21,     31, 5.98    , -55731.43
  data     20,    -10, 4.03    ,  60697.74
  data     18,      0, 4.27    ,   2132.79
  data     17,    -12, 0.79    , 109771.63
  data     14,      0, 4.24    ,  -7752.82
  data     13,     -5, 2.01    , 188491.91
  data     13,      0, 2.65    ,    207.81
  data     13,      0, 4.98    ,  29424.63
  data     12,      0, 0.93    ,     -7.99
  data     10,      0, 2.21    ,  46941.14
  data     10,      0, 3.59    ,    -68.29
  data     10,      0, 1.50    ,  21463.25
  data     10,     -9, 2.55    , 157208.40
  ' read leap second data
  for i% = 1 to 28
    read jdleap(i%), leapsec(i%)
  next i%
  data 2441317.5,  10.0
  data 2441499.5,  11.0
  data 2441683.5,  12.0
  data 2442048.5,  13.0
  data 2442413.5,  14.0
  data 2442778.5,  15.0
  data 2443144.5,  16.0
  data 2443509.5,  17.0
  data 2443874.5,  18.0
  data 2444239.5,  19.0
  data 2444786.5,  20.0
  data 2445151.5,  21.0
  data 2445516.5,  22.0
  data 2446247.5,  23.0
  data 2447161.5,  24.0
  data 2447892.5,  25.0
  data 2448257.5,  26.0
  data 2448804.5,  27.0
  data 2449169.5,  28.0
  data 2449534.5,  29.0
  data 2450083.5,  30.0
  data 2450630.5,  31.0
  data 2451179.5,  32.0
  data 2453736.5,  33.0
  data 2454832.5,  34.0
  DATA 2456109.5,  35.0
  data 2457204.5,  36.0
  data 2457754.5,  37.0
  ' calendar months
  month$(1) = "January"
  month$(2) = "February"
```

```
month$(3) = "March"
month$(4) = "April"
month$(5) = "May"
month$(6) = "June"
month$(7) = "July"
month$(8) = "August"
month$(9) = "September"
month$(10) = "October"
month$(11) = "November"
month$(12) = "December"
''''''''''''''''''
' begin simulation
''''''''''''''''''
print " "
print "time of the seasons"
print "==================="
print " "
' request calendar year
print "please input the calendar year"
input year
' process each season
for iequsol% = 1 to 4
  select case iequsol%
    case (1)
      cmonth = 3
      day = 15
      julian(cmonth, day, year, jdayi)
    case (2)
      cmonth = 6
      day = 15
      julian(cmonth, day, year, jdayi)
      along2 = 0.5 * pi
    case (3)
      cmonth = 9
      day = 15
      julian(cmonth, day, year, jdayi)
    case (4)
      cmonth = 12
      day = 15
      julian(cmonth, day, year, jdayi)
      along2 = 1.5 * pi
  end select
  ' find event
  x1 = 0.0
  x2 = 10.0
  realroot1(x1, x2, 1.0e-8, xroot, froot)
  ' TDB julian day of event
  jdtdb = jdayi + xroot
  ' compute UTC julian day
  tdb2utc(jdtdb, jdutc)
  ' print results for this event
```

```
    print " "
    select case iequsol%
      case (1)
        print "spring equinox"
        print "--------------"
      case (2)
        print "summer solstice"
        print "---------------"
      case (3)
        print "fall equinox"
        print "------------"
      case (4)
        print "winter solstice"
        print "---------------"
    end select
    print " "
    gdate (jdutc, cmonth, day, year)
    print "calendar date  ", month$(cmonth), " ", STR$(int(day)), " ",
str$(year)
    print " "
    thr0 = 24.0 * (day - int(day))
    thr = int(thr0)
    tmin0 = 60.0 * (thr0 - thr)
    tmin = int(tmin0)
    tsec = 60.0 * (tmin0 - tmin)
    print "UTC time       ", str$(thr) + " hours " + str$(tmin) + " minutes
" + str$(tsec, 0, 2) + " seconds"
  next iequsol%
  print " "
end
  ''''''''''''''''
  ''''''''''''''''
sub esfunc(x, fx)
  ' equinox/solstice objective function
  ''''''''''''''''''''''''''''''''''''''
  local jday, rlsun, rasc, decl
  jday = jdayi + x
  solar(jday, rlsun, rasc, decl)
  if (iequsol% = 1 or iequsol% = 3) then
    fx = decl
  else
    fx = along2 - rlsun
  end if
end sub
  ''''''''''''''''''''''''
  ''''''''''''''''''''''''
sub tdb2utc (jdtdb, jdutc)
  ' convert TDB julian day to UTC julian day subroutine
  ' input
  '  jdtdb = TDB julian day
```

```
  ' output
  '  jdutc = UTC julian day
  ''''''''''''''''''''''''''
  local x1, x2, xroot, froot
  jdsaved = jdtdb
  ' set lower and upper bounds
  x1 = jdsaved - 0.1
  x2 = jdsaved + 0.1
  ' solve for UTC julian day using Brent's method
  realroot2(x1, x2, 1.0e-8, xroot, froot)
  jdutc = xroot
end sub
  ''''''''''''''''''
  ''''''''''''''''''
sub jdfunc (jdin, fx)
  ' objective function for tdb2utc
  ' input
  '  jdin = current value for UTC julian day
  ' output
  '  fx = delta julian day
  ''''''''''''''''''''''''
  local jdwrk, tai_utc
  findleap(jdin, tai_utc)
  utc2tdb(jdin, tai_utc, jdwrk)
  fx = jdwrk - jdsaved
end sub
  ''''''''''''''''''''''''''''''
  ''''''''''''''''''''''''''''''
sub solar (jd, rlsun, rasc, decl)
  ' precision ephemeris of the Sun
  ' input
  '  jd = julian ephemeris day
  ' output
  '  rlsun = ecliptic longitude of the sun
  '          (0 <= rlsun <= 2 pi)
  '  rasc  = right ascension of the Sun (radians)
  '          (0 <= rasc <= 2 pi)
  '  decl  = declination of the Sun (radians)
  '          (-pi/2 <= decl <= pi/2)
  ''''''''''''''''''''''''''''''''''
  local u, a1, a2, psi, deps, meps, eps, seps, ceps
  local dl, dr, w, srl, crl, srb, crb, sra, cra
  u = (jd - 2451545.0) / 3652500.0
  ' compute nutation in longitude
  a1 = 2.18 + u * (-3375.7 + u * 0.36)
  a2 = 3.51 + u * (125666.39 + u * 0.1)
  psi = 0.0000001 * (-834.0 * sin(a1) - 64.0 * sin(a2))
  ' compute nutation in obliquity
  deps = 0.0000001 * u * (-226938 + u * (-75 + u * (96926 + u * (-2491 - u *
12104))))
  meps = 0.0000001 * (4090928.0 + 446.0 * cos(a1) + 28.0 * cos(a2))
```

```
  eps = meps + deps
  seps = sin(eps)
  ceps = cos(eps)
  dl = 0.0
  dr = 0.0
  for i% = 1 to 50
    w = sa(i%) + sb(i%) * u
    dl = dl + sl(i%) * sin(w)
    if (sr(i%) <> 0.0) then
      dr = dr + sr(i%) * cos(w)
    end if
  next i%
  dl = modulo(dl * 0.0000001 + 4.9353929 + 62833.196168 * u)
  dr = 149597870.691 * (dr * 0.0000001 + 1.0001026)
  rlsun = modulo(dl + 0.0000001 * (-993.0 + 17.0 * cos(3.1 + 62830.14 * u))
+ psi)
  rb = 0.0
  ' compute geocentric declination and right ascension
  crl = cos(rlsun)
  srl = sin(rlsun)
  crb = cos(rb)
  srb = sin(rb)
  decl = asin(ceps * srb + seps * crb * srl)
  sra = -seps * srb + ceps * crb * srl
  cra = crb * crl
  rasc = atan3(sra, cra)
end sub
  '''''''''''''''''''''''''''''''''''''''
  '''''''''''''''''''''''''''''''''''''''
sub realroot1(x1, x2, tol, xroot, froot)
  ' real root of a single non-linear function subroutine
  ' input
  '  x1  = lower bound of search interval
  '  x2  = upper bound of search interval
  '  tol = convergence criter%ia
  ' output
  '  xroot = real root of f(x) = 0
  '  froot = function value
  ' note: requires sub esfunc
  '''''''''''''''''''''''''''''
  local eps, a, b, c, d, e, fa, fb, fcc, tol1
  local xm, p, q, r, s, xmin, tmp
  eps = 2.23e-16
  e = 0.0
  a = x1
  b = x2
  esfunc(a, fa)
  esfunc(b, fb)
  fcc = fb
  for iter% = 1 to 50
```

```
    if (fb * fcc > 0.0) then
      c = a
      fcc = fa
      d = b - a
      e = d
    end if
    if (abs(fcc) < abs(fb)) then
      a = b
      b = c
      c = a
      fa = fb
      fb = fcc
      fcc = fa
    end if
    tol1 = 2.0 * eps * abs(b) + 0.5 * tol
    xm = 0.5 * (c - b)
    if (abs(xm) <= tol1 or fb = 0.0) then exit for
    if (abs(e) >= tol1 and abs(fa) > abs(fb)) then
      s = fb / fa
      if (a = c) then
        p = 2.0 * xm * s
        q = 1.0 - s
      else
        q = fa / fcc
        r = fb / fcc
        p = s * (2.0 * xm * q * (q - r) - (b - a) * (r - 1.0))
        q = (q - 1.0) * (r - 1.0) * (s - 1.0)
      end if
      if (p > 0.0) then q = -q
      p = abs(p)
      min = abs(e * q)
      tmp = 3.0 * xm * q - abs(tol1 * q)
      if (min < tmp) then min = tmp
      if (2.0 * p < min) then
        e = d
        d = p / q
      else
        d = xm
        e = d
      end if
    else
      d = xm
      e = d
    end if
    a = b
    fa = fb
    if (abs(d) > tol1) then
      b = b + d
    else
      b = b + sgn(xm) * tol1
    end if
```

```
    esfunc(b, fb)
  next iter%
  froot = fb
  xroot = b
end sub
  '''''''''''''''''''''''''''''''''''''''
  '''''''''''''''''''''''''''''''''''''''
sub realroot2(x1, x2, tol, xroot, froot)
  ' real root of a single non-linear function subroutine
  ' input
  '  x1  = lower bound of search interval
  '  x2  = upper bound of search interval
  '  tol = convergence criter%ia
  ' output
  '  xroot = real root of f(x) = 0
  '  froot = function value
  ' note: requires sub jdfunc
  '''''''''''''''''''''''''''''
  local eps, a, b, c, d, e, fa, fb, fcc, tol1
  local xm, p, q, r, s, xmin, tmp
  eps = 2.23e-16
  e = 0.0
  a = x1
  b = x2
  jdfunc(a, fa)
  jdfunc(b, fb)
  fcc = fb
  for iter% = 1 to 50
    if (fb * fcc > 0.0) then
      c = a
      fcc = fa
      d = b - a
      e = d
    end if
    if (abs(fcc) < abs(fb)) then
      a = b
      b = c
      c = a
      fa = fb
      fb = fcc
      fcc = fa
    end if
    tol1 = 2.0 * eps * abs(b) + 0.5 * tol
    xm = 0.5 * (c - b)
    if (abs(xm) <= tol1 or fb = 0) then exit for
    if (abs(e) >= tol1 and abs(fa) > abs(fb)) then
      s = fb / fa
      if (a = c) then
        p = 2.0 * xm * s
        q = 1.0 - s
```

```
      else
        q = fa / fcc
        r = fb / fcc
        p = s * (2.0 * xm * q * (q - r) - (b - a) * (r - 1.0))
        q = (q - 1.0) * (r - 1.0) * (s - 1.0)
      end if
      if (p > 0) then q = -q
      p = abs(p)
      xmin = abs(e * q)
      tmp = 3.0 * xm * q - abs(tol1 * q)
      if (xmin < tmp) then xmin = tmp
      if (2.0 * p < xmin) then
        e = d
        d = p / q
      else
        d = xm
        e = d
      end if
    else
      d = xm
      e = d
    end if
    a = b
    fa = fb
    if (abs(d) > tol1) then
      b = b + d
    else
      b = b + sgn(xm) * tol1
    end if
    jdfunc(b, fb)
  next iter%
  froot = fb
  xroot = b
end sub
  ''''''''''''''''''''''''''''''''
  ''''''''''''''''''''''''''''''''
sub julian(month, day, year, jday)
  ' Gregorian date to julian day subroutine
  ' input
  '  month = calendar month
  '  day   = calendar day
  '  year  = calendar year (all four digits)
  ' output
  '  jday = julian day
  ' special notes
  '  (1) calendar year must include all digits
  '  (2) will report October 5, 1582 to October 14, 1582
  '      as invalid calendar dates and exit
  ''''''''''''''''''''''''''''''''''''''''''
  local a, b, c, m, y
  y = year
```

```
  m = month
  b = 0.0
  c = 0.0
  if (m <= 2.0) then
    y = y - 1.0
    m = m + 12.0
  end if
  if (y < 0.0) then c = -0.75
  if (year < 1582.0) then
    ' null
  elseif (year > 1582.0) then
    a = fix(y / 100.0)
    b = 2.0 - a + fix(a / 4.0)
  elseif (month < 10.0) then
    ' null
  elseif (month > 10.0) then
    a = fix(y / 100.0)
    b = 2.0 - a + fix(a / 4.0)
  elseif (day <= 4.0) then
    ' null
  elseif (day > 14.0) then
    a = fix(y / 100.0)
    b = 2.0 - a + fix(a / 4.0)
  else
    print "this date does not exist!!"
    exit
  end if
  jday = fix(365.25 * y + c) + fix(30.6001 * (m + 1.0)) + day + b +
1720994.5
end sub
  '''''''''''''''''''''''''''''''''
  '''''''''''''''''''''''''''''''''
sub gdate (jday, month, day, year)
  ' Julian day to Gregorian date subroutine
  ' input
  '  jday = julian day
  ' output
  '  month = calendar month
  '  day   = calendar day
  '  year  = calendar year
  ''''''''''''''''''''''''''
  local a, b, c, d, e, f, z, alpha
  z = fix(jday + 0.5)
  f = jday + 0.5 - z
  if (z < 2299161) then
    a = z
  else
    alpha = fix((z - 1867216.25) / 36524.25)
    a = z + 1.0 + alpha - fix(alpha / 4.0)
  end if
```

```
  b = a + 1524.0
  c = fix((b - 122.1) / 365.25)
  d = fix(365.25 * c)
  e = fix((b - d) / 30.6001)
  day = b - d - fix(30.6001 * e) + f
  if (e < 13.5) then
    month = e - 1.0
  else
    month = e - 13.0
  end if
  if (month > 2.5) then
    year = c - 4716.0
  else
    year = c - 4715.0
  end if
end sub
  ''''''''''''''''''''''''''''''''''
  ''''''''''''''''''''''''''''''''''
sub utc2tdb (jdutc, tai_utc, jdtdb)
  ' convert UTC julian date to TDB julian date
  ' input
  '  jdutc   = UTC julian day
  '  tai_utc = TAI-UTC (seconds)
  ' output
  '  jdtdb = TDB julian day
  ' Reference Frames in Astronomy and Geophysics
  ' J. Kovalevsky et al., 1989, pp. 439-442
  ''''''''''''''''''''''''''''''''''''''''''
  local corr, jdtdt, t
  ' TDT julian date
  corr = (tai_utc + 32.184) / 86400.0
  jdtdt = jdutc + corr
  ' time argument for correction
  t = (jdtdt - 2451545.0) / 36525.0
  ' compute correction in microseconds
  corr = 1656.675 * sin(dtr * (35999.3729 * t + 357.5287))
  corr = corr + 22.418     * sin(dtr * (32964.467  * t + 246.199))
  corr = corr + 13.84      * sin(dtr * (71998.746  * t + 355.057))
  corr = corr +  4.77      * sin(dtr * ( 3034.906  * t +  25.463))
  corr = corr +  4.677     * sin(dtr * (34777.259  * t + 230.394))
  corr = corr + 10.216 * t * sin(dtr * (35999.373  * t + 243.451))
  corr = corr +  0.171 * t * sin(dtr * (71998.746  * t + 240.98 ))
  corr = corr +  0.027 * t * sin(dtr * ( 1222.114  * t + 194.661))
  corr = corr +  0.027 * t * sin(dtr * ( 3034.906  * t + 336.061))
  corr = corr +  0.026 * t * sin(dtr * (  -20.186  * t +   9.382))
  corr = corr +  0.007 * t * sin(dtr * (29929.562  * t + 264.911))
  corr = corr +  0.006 * t * sin(dtr * (  150.678  * t +  59.775))
  corr = corr +  0.005 * t * sin(dtr * ( 9037.513  * t + 256.025))
  corr = corr +  0.043 * t * sin(dtr * (35999.373  * t + 151.121))
  ' convert corrections to days
  corr = 0.000001 * corr / 86400.0
```

```
  ' TDB julian date
  jdtdb = jdtdt + corr
end sub
  '''''''''''''''''''''''''''''
  '''''''''''''''''''''''''''''
sub findleap(jday, leapsecond)
  ' find number of leap seconds for utc julian day
  ' input
  '  jday = utc julian day
  ' input via global
  '  jdleap  = array of utc julian dates
  '  leapsec = array of leap seconds
  ' output
  '  leapsecond = number of leap seconds
  '''''''''''''''''''''''''''''''''''''''''''
  if (jday <= jdleap(1)) then
    ' date is <= 1972; set to first data element
    leapsecond = leapsec(1)
    exit sub
  end if
  if (jday >= jdleap(28)) then
    ' date is >= end of current data
    ' set to last data element
    leapsecond = leapsec(28)
    exit sub
  end if
  ' find data within table
  for i% = 1 to 27
    if (jday >= jdleap(i%) and jday < jdleap(i% + 1)) then
      leapsecond = leapsec(i%)
      exit sub
    end if
  next i%
end sub
  ''''''''''''''''''''''''''''
  ''''''''''''''''''''''''''''
function modulo(x) as float
  ' modulo 2 pi function
  ''''''''''''''''''''''''''
  local a
  a = x - pi2 * fix(x / pi2)
  if (a < 0.0) then
    a = a + pi2
  end if
  modulo = a
end function
  '''''''''''''''''''''''''''''
  '''''''''''''''''''''''''''''
function atan3(a, b) as float
  ' four quadrant inverse tangent function
```

```
  ' input
  '  a = sine of angle
  '  b = cosine of angle
  ' output
  '  atan3 = angle (0 =< atan3 <= 2 * pi; radians)
  ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
  local c
  if (abs(a) < 1.0e-10) then
    atan3 = (1.0 - sgn(b)) * pidiv2
    exit function
  else
    c = (2.0 - sgn(a)) * pidiv2
  endif
  if (abs(b) < 1.0e-10) then
    atan3 = c
    exit function
  else
    atan3 = c + sgn(a) * sgn(b) * (abs(atn(a / b)) - pidiv2)
  endif
end function
```