```
//  The below routines convert from X, Y, Z to Lat/Lon/Alt and the reverse.
//  These functions assume the WGS-84 reference system and do not accommodate
//  other datums.
//  For reference, The X, Y, Z coordinate system has origin at the mass center
//  of the Earth, with Z axis along the spin axis of the Earth (+ at North pole).
//  The +X axis emerges from the Earth at the equator-prime meridian intersection.
//  The +Y axis defines a right-hand coordinate system, emerging from the Earth
//  at +90 degrees longitude on the equator.
//
//  The Local Tangential Plane (LTP) coordinates are in latitude/longitude/altitude
//  where North latitude is + and East longitude is +.  Altitude is in meters above
//  the reference ellipsoid (which may be either above or below the local mean sea
//  level).

//  Note: the below code was extracted from working software, but has been edited
//  for this document.  If any problems occur in using this, please contact the
//  engineer who provided it to you for assistance.

//  Define some earth constants
#define MAJA   (6378137.0)             // semi major axis of ref ellipsoid
#define FLAT   (1.0/298.2572235630)    // flattening coef of ref ellipsoid.
//  These are derived values from the above WGS-84 values
#define ESQR   (FLAT * (2.0-FLAT))     // 1st eccentricity squared
#define OMES   (1.0 - ESQR)            // 1 minus eccentricity squared
#define EFOR   (ESQR * ESQR)           // Sqr of the 1st eccentricity squared
#define ASQR   (MAJA * MAJA)           // semi major axis squared = nlmaja**


void ConvertECEFToLTP(double nlecef[3],
                      double * nllat,
                      double * nllon,
                      double * nlalt )
{  //  Convert from ECEF (XYZ) to LTP (lat/lon/alt)

   double     nla0,nla1,nla2,nla3,nla4,nlb0,nlb1,nlb2,nlb3;
   double     nlb,nlc0,nlopqk,nlqk,nlqkc,nlqks,nlf,nlfprm;
   long int   nlk;
   double     ytemp, xtemp;

   /*-------------------------------------------*/
   /* b = (x*x + y*y) / semi_major_axis_squared */
   /* c = (z*z) / semi_major_axis_squared       */
   /*-------------------------------------------*/

   nlb = (nlecef[0] * nlecef[0] + nlecef[1] * nlecef[1]) / ASQR;
   nlc0 = nlecef[2] * nlecef[2] / ASQR;

   /*-------------------------------------------*/
   /* a0 = c * one_minus_eccentricity_sqr       */
   /* a1 = 2 * a0                               */
   /* a2 = a0 + b - first_eccentricity_to_fourth*/
   /* a3 = -2.0 * first_eccentricity_to_fourth  */
   /* a4 = - first_eccentricity_to_fourth       */
   /*-------------------------------------------*/

   nla0 = OMES * nlc0;
   nla1 = 2.0 * nla0;
```

```
nla2 = nla0 + nlb - EFOR;
nla3 = -2.0 * EFOR;
nla4 = - EFOR;

/*-------------------------------------------*/
/* b0 = 4 * a0, b1 = 3 * a1                  */
/* b2 = 2 * a2, b3 = a3                      */
/*-------------------------------------------*/

nlb0 = 4.0 * nla0;
nlb1 = 3.0 * nla1;
nlb2 = 2.0 * nla2;
nlb3 = nla3;

/*-------------------------------------------*/
/* Compute First Eccentricity Squared        */
/*-------------------------------------------*/

nlqk = ESQR;

for(nlk = 1; nlk <= 3; nlk++)
{
   nlqks = nlqk * nlqk;
   nlqkc = nlqk * nlqks;
   nlf   = nla0 * nlqks * nlqks + nla1 * nlqkc + nla2 * nlqks +
           nla3 * nlqk + nla4;
   nlfprm= nlb0 * nlqkc + nlb1 * nlqks + nlb2 * nlqk + nlb3;
   nlqk  = nlqk - (nlf / nlfprm);
}

/*-------------------------------------------*/
/* Compute latitude, longitude, altitude  */
/*-------------------------------------------*/

nlopqk = 1.0 + nlqk;

if ( nlecef[0] == 0.0 && nlecef[1] == 0.0 )/* on the earth's axis */
{
   /* We are sitting EXACTLY on the earth's axis */
   /* Probably at the center or on or near one of the poles */
   *nllon = 0.0;                        /* as good as any other value */
   if(nlecef[2] >= 0.0)
      *nllat = PI / 2;                  /* alt above north pole */
   else
      *nllat = -PI / 2;                 /* alt above south pole */
}
else
{
   ytemp = nlopqk * nlecef[2];
   xtemp = sqrt(nlecef[0] * nlecef[0] + nlecef[1] * nlecef[1]);

   *nllat = atan2(ytemp, xtemp);
   *nllon = atan2(nlecef[1], nlecef[0]);
}

*nlalt = (1.0 - OMES / ESQR * nlqk) * MAJA *
         sqrt(nlb / (nlopqk * nlopqk) + nlc0);
```

```
}  // ConvertECEFToLTP()


typedef struct
{
   DOUBLE  Lat;  //  in radians, + = North
   DOUBLE  Lon;  //  in radians, + = East
   DOUBLE  Alt;  //  in meters above the reference ellipsoid
} LTP;

typedef struct
{  //  Earth-Centered, Earth-Fixed in WGS-84 system
   DOUBLE  X;  //  all values are in meters
   DOUBLE  Y;
   DOUBLE  Z;
} ECEF;

void ConvertLTPToECEF(LTP *plla, ECEF *pecef)
{  //  assumes input is in WGS-84, output is also in WGS-84
   double slat, clat, r;

   slat = sin(plla->Lat);
   clat = cos(plla->Lat);

   r = MAJA / sqrt(1.0 - ESQR * slat * slat);

   pecef->X = (r + plla->Alt) * clat * cos(plla->Lon);
   pecef->Y = (r + plla->Alt) * clat * sin(plla->Lon);
   pecef->Z = (r * OMES + plla->Alt) * slat;

   return;

}  //  ConvertLTPToECEF ()




// Given a current lat/lon, compute the direction cosine matrix cen[3][3].  Then, to
// compute the updated velocity N, E, D from either current X,Y,Z or velocity(X,Y or Z),
// use the matrix as follows:

// Velocity(North) = cen[0][0] * velocity(x) +
//                   cen[0][1] * velocity(y) +
//                   cen[0][2] * velocity(z);
// Velocity(East)  = cen[1][0] * velocity(x) +
//                   cen[1][1] * velocity(y) +
//                   cen[1][2] * velocity(z);
// Velocity(Down)  = cen[2][0] * velocity(x) +
//                   cen[2][1] * velocity(y) +
//                   cen[2][2] * velocity(z);



void E2NDCM (double lat, double lon, double cen[][3])
```

```
{
//
//
//  Row CEN[0][i] is the North Unit vector in ECEF
//  Row CEN[1][i] is the East Unit vector
//  Row CEN[2][i] is the Down Unit vector
//
//

   double slat, clat, slon, clon;

   slat = sin (lat);
   clat = cos (lat);
   slon = sin (lon);
   clon = cos (lon);

   cen[0][0] = -clon * slat;
   cen[0][1] = -slon * slat;
   cen[0][2] = clat;

   cen[1][0] = -slon;
   cen[1][1] = clon;
   cen[1][2] = 0.0;

   cen[2][0] = -clon * clat;
   cen[2][1] = -slon * clat;
   cen[2][2] = -slat;

   return;

}  //  end E2NDCM ()
```