

/\*

**EPHEMERIS,  
Introduction to planetary ephemeris calculation  
Una introducción al cálculo de efemérides planetarias**

**Juan Lacruz**

**This document may be freely distributed unchanged  
Este documento puede ser libremente distribuido sin cambios**

**May, 1993,  
1<sup>st</sup> Revision 2002**

---

Basic astronomy, Julian date, sidereal time  
Position of the Sun, planets, asteroids and  
Elliptical orbit comets

This document is a C program that can be compiled  
and executed to obtain ephemeris of solar system bodies.

---

Astronomía básica, día Juliano, tiempo sidéreo  
Posiciones del Sol, planetas, asteroides y  
Cometas de órbita elíptica

Este documento es un programa C que puede ser compilado  
Y ejecutado para obtener efemérides de objetos del  
Sistema Solar.

---

**Bibliography :**

**TEODORO J.VIVES**  
**Astronomia de posicion**  
**Ed. Alhambra, Madrid 1971 col. EXEDRA**

**DANJON**  
**Astronomie general**  
**Ed. Sennac, Paris 1959**

\*/

```
#include <stdio.h>  
#include <math.h>
```

```
main()
```

```

{
/*
    Variable definitions
    Definiciones de variables
*/

    int  year, month, day;
    int  grado, hour, min, sec;

/*
    Longitude is + to the East and - to the west of Greenwich
    Madrid's longitud aprox. -3.8 grados

    Longitud + al este y – al oeste de Greenwich
    La longitud de Madrid es aprox. –3.8 grados
*/

    float longitud=-3.8;
    float julian;
    float jdsince1900;

    double t, tsg;
    double L, E, M, E1;
    double alfa, alfarad, delta;
    double v, r;
    double X, Y, Z, x, y, z;
    double PX, PY, PZ, QX, QY, QZ;
    double GI, ETA, ZETA;

/*
    Constant definition
    Definición de constantes

    n = mean solar movement in arc seconds per day.
        movimiento solar medio en segundos de arco por día.

jd1900jan0 = 1900 JAN 0 Julian date
    e = Ecliptical obliquity; oblicuidad de la eclíptica.
    epsilon = Earth's orbit eccentricity; excentricidad de la órbita terrestre
*/

    double jd1900jan0 = 2415020.0;
    double n           = 3548.19281;
    double pi          = 3.1415926535;
    double e           = 23.45;
    double epsilon     = 0.01671143;
/*

```

Orbital parameters for the object to compute.  
Parámetros orbitales del objeto a calcular.

\*/

```
double omega, i, w, a, exc, t0;
```

```
printf("\n Calculation of the Julian date, sidereal time,\n");  
printf("\n Sun's and various solar system bodies positions.");  
printf("\n valid for Gregorian dates since 1901 to 2100\n\n\n");
```

```
printf("\n Cálculo del día Juliano, tiempo sidéreo,\n");  
printf("\n Posición del Sol y objetos del sistema Solar.");  
printf("\n válido para fechas Gregorianas desde 1901 a 2100\n\n\n");
```

```
printf(" Longitude degrees (-)West, (+)East : ");  
scanf("%f",&longitud);
```

```
printf("\n\n");  
printf(" Year YYYY :");  
scanf("%d", &year);
```

```
printf(" Month MM :");  
scanf("%d", &month);
```

```
printf(" Day DD :");  
scanf("%d", &day);
```

```
printf("\n\n");  
printf(" GMT Hour HH :");  
scanf("%d",&hour);
```

```
printf(" Minutes MM :");  
scanf("%d",&min);
```

```
printf(" Seconds SS :");  
scanf("%d",&sec);
```

/\*

First thing, calculate the Julian date at 0h GMT at Greenwich  
Lo primero, calcular el día Juliano a 0h GMT en Greenwich

\*/

```
julian = (4712+year)*365.25;
```

```
if (julian == floor(julian))
```

```
    julian = julian-1;
else
    julian = floor(julian);
```

```
/*
```

```
Account for the Gregorian modification of the calendar by subtracting 13 days
Tener en cuenta la modificación del calendario por el Papa Gregorio restando 13 días
```

```
*/
```

```
    julian=julian-13;
```

```
/*
```

```
Find the day of the year
Calcular el día del año
```

```
*/
```

```
    julian=julian + day;
```

```
    if (month > 1)
        julian=julian+31;
    if (month > 2)
        julian=julian+28;
    if (month > 3)
        julian=julian+31;
    if (month > 4)
        julian=julian+30;
    if (month > 5)
        julian=julian+31;
    if (month > 6)
        julian=julian+30;
    if (month > 7)
        julian=julian+31;
    if (month > 8)
        julian=julian+31;
    if (month > 9)
        julian=julian+30;
    if (month > 10)
        julian=julian+31;
    if (month > 11)
        julian=julian+30;
```

```
/*    If it's a lap year february is one day longer. */
```

```
/*    En los bisiestos febrero tiene un día más    */
```

```
if(year == floor(year/4)*4 & month > 1)
    julian=julian + 1;
```

```
/*
```

Subtract half a day to account for the old difference between the beginnings of the civil and astronomical days.

Restar medio día para tener en cuenta la antigua Diferencia entre los comienzos del día civil y El día astronómico.

```
*/
```

```
julian=julian - .5;  
jdsince1900=julian-jd1900jan0;
```

```
printf("\n Julian date %f\n", julian);  
printf(" since 1900 jan 0 %f\n",jdsince1900);
```

```
/*
```

To calculate the sidereal time fixing the position of the Celestial sphere(aries point) for a given instant, Newcomb's formulae is used, in which t is the number of julian centuries, of 36525 mean solar days, elapsed since 1900 JAN 0

Para calcular el tiempo sidéreo que fija la posición de ls bóveda Celeste (punto aries) en un momento dado, se utiliza la fórmula de Newcomb, en la cual t es el número de siglos Julianos de 36525 días medios, transcurridos desde 1900 ENE 0

```
*/
```

```
t=jdsince1900/36525;  
tsg=6.64606 + 2400.05126 * t + 2.58055e-5 * t * t;
```

```
/*
```

To get the local sidereal time, the longitude must be considered.  
Para obtener el tiempo sidéreo local, se debe corregir por longitud.

```
*/
```

```
tsg=tsg+longitud/15.0;
```

```
/*
```

Now we have to add the fraction of day, as a sideral time interval, elapsed since 0 h GMT.

To convert from UT to ST we multiply by the number of sidereal days a mean solar day has, aprox :

Ahora tenemos que sumar la fracción de día, como intervalo de tiempo sidéreo, que ha transcurrido desde 0h GMT.

Para convertir de TU a TS se multiplica por el número de días sidéreos que hay en un día solar medio, aproximadamente :

(24h 3 min / 24 h)

\*/

```
tsg = tsg + (hour+min/60.0+sec/3600.0)*1.002737909265;
```

```
t = tsg - floor(tsg/24)*24;
```

```
hour = floor(t);
```

```
min = floor((t-hour)*60);
```

```
sec = floor((t - hour - min/60.0)*3600);
```

```
printf("\n Local sidereal time %d h, %d m, %d s", hour, min, sec);
```

/\*

Now the Sun's position is computed, to that end, the time equation E has to be found in seconds of time.

Ahora se calcula la posición del Sol, para ello, se calcula la ecuación del tiempo en segundos de tiempo.

$$E=459.912 \sin (M) - 591.976 \sin (2 * L) - 39.609 \sin (M) \cos (2 * L)$$

Where M is the Sun's mean anomaly and L is the Sun's mean Longitude.

Donde M es la anomalía Solar media y L es la longitud Solar media.

According to Danjon :

Según Danjon :

$$L=1006908.04+129602768.13 * t+1.089 * t * t \text{ (arc seconds)}$$

$$M=n(t-t_0)$$

Where  $t_0$  is a perihelion date. In this example to take that of 1950 January 3

$$t_0=2433284.5$$

Donde  $t_0$  es una fecha de paso por el perihelio, en este ejemplo tomamos la de 1950 enero 3

$t_0=2433284.5$

t is the number of julian centuries of 36525 mean days elapsed since 1900 JAN 0.

t es el número de siglos Julianos de 36525 días solares medios transcurridos desde 1900 enero 0

\*/

```
t=jdsince1900/36525;  
L=(1006908.04+129602768.13*t+1.089*t*t)/3600;
```

```
L=L-floor(L/360)*360;  
printf("\n Sun's mean longitude (degrees) %f",L);  
L=L*pi/180;
```

```
M=n*(julian-2433284.5)/3600;
```

```
M=M-floor(M/360)*360;
```

```
printf("\n Sun's mean anomaly (degrees) %f",M);
```

```
M=M*pi/180;
```

```
alfa=67125.836+8640184.542*t+0.0929*t*t;
```

```
alfarad=alfa/60/60*360/24*pi/180;
```

```
E=459.912*sin(M) - 591.976*sin(2*alfarad) - 9.609*sin(M)*cos(2*alfarad);  
E=E+12.744*sin(4*alfarad)+4.807*sin(2*M);  
E=E/60;
```

```
min = floor(E);  
sec = floor((E-min)*60);
```

```
printf("\n Equation of time (0h TU) %d m, %d s", min, sec);
```

/\*

Sun's movement in right ascension :  
True Sun's right ascension

Movimiento del Sol en ascensión recta :  
Ascensión recta del Sol verdadero

\*/

```
alfa=alfa+E*60;
```

```
alfa=alfa*360/60/60/24;  
alfa=alfa-floor(alfa/360)*360;  
alfarad=alfa*pi/180;  
alfa=alfa*24/360;
```

```
hour = floor(alfa);  
min = floor((alfa-hour)*60);  
sec = floor((alfa - hour - min/60.0)*3600);
```

```
printf("\n True Sun's right ascension %d h, %d m, %d s", hour, min, sec);
```

```
/*
```

```
True Sun's longitude (degrees)  
Longitud del Sol verdadero (grados)
```

```
*/
```

```
L=282.075+M*180/pi+1.916666*sin(M);  
L=L-floor(L/360)*360;
```

```
printf("\n True Sun's longitude %f deg.", L);
```

```
/*
```

```
Vector radius  
Radio vector
```

```
*/
```

```
r=1.00013995 - 0.01672838*cos(M)-0.00013992*cos(2*M);  
r=r-0.00000176*cos(3*M)-0.00000003*cos(4*M);
```

```
printf("\n Sun's vector radius %f U.A.", r);
```

```
/*
```

```
Sun's geocentric ecliptical rectangular coordinates.  
Coordenadas rectangulares eclípticas geocéntricas del Sol
```

```
*/
```

```
X=r*cos(L*pi/180);  
Y=r*sin(L*pi/180);
```

```
Z=0;
printf("\n\n Sun's geocentric ecliptical rectangular coordinates");
printf("\n\n X= %f Y= %f Z= %f", X, Y, Z);
```

```
/*
```

```
Sun's geocentric ecuatorial rectangular coordinates
Coordenadas rectangulares ecuatoriales geocéntricas del Sol
```

```
*/
```

```
X=r*cos(L*pi/180);
Y=r*sin(L*pi/180)*cos(e*pi/180);
Z=r*sin(L*pi/180)*sin(e*pi/180);

printf("\n\n Sun's geocentric ecuatorial rectangular coordinates");
printf("\n\n X= %f Y= %f Z= %f", X, Y, Z);
```

```
/*
```

```
Objects's orbit data
Datos de la órbita del objeto
```

Omega=	Ascending node longitude	degrees
i	= Orbit inclination	degrees
w	= Perihelion argument	degrees
a	= Semimajor axis	A.U.
exc	= eccentricity	
t0	= Perihelion epoch	julian day
n	= Mean movement	degrees/day

omega	= Longitud del nodo ascendente	grados
i	= Inclinación de la órbita	grados
w	= Argumento del perihelio	grados
a	= Semieje mayor	A.U.
exc	= excentricidad	
t0	= Época de paso por el perihelio	julian day
n	= Movimiento medio	degrees/day

```
*/
```

```
/* Comet Oterma (no. 804 Catalogue of cometary orbits) */
```

```
/*
```

```
omega=155.1100;
i =3.9921;
w =354.8723;
```

```

a =3.960045;
exc =0.144497;
n =0.125070;
t0 =2436365.0045;
*/
/* Comet Encke */
/*
    omega=334.8;
    i =11.9;
    w =186.2;
    a =2.207;
    exc =0.850;
    n =0.3005;
    t0 =2448191.5;
*/
/* Comet Kojima */
/*
    omega=154.7;
    i =0.9;
    w =348.5;
    a =1.6420;
    exc =0.391;
    n =0.1249;
    t0 =2446525.5;
*/
/* Asteroid Vesta */

    omega=103.974;
    i =7.135;
    w =150.183;
    a =2.3616;
    exc =0.0896;
    n =1/pow(a,1.5)*360/365.25;
    t0 =2448968.4875;

M=n*(julian-t0);

M=M-floor(M/360)*360;

printf("\n  Object's mean anomaly (degrees) %f",M);

E1=0;
E=M+exc*sin(M*pi/180)*180/pi;

while (abs(E1-E)>0.00001) {
    E1=E;
    E=M+exc*sin(E*pi/180)*180/pi;
}

```

```

printf("\n  Object's eccentric anomaly (degrees) %f",E);

v=sqrt((1+exc)/(1-exc))*tan(E/2*pi/180);
v=2*atan(v)*180/pi;

if (v<0) v=v+360;

printf("\n  Object's true anomaly (degrees) %f",v);

r=a*(1-exc*exc)/(1+exc*cos(v*pi/180));
printf("\n  Object's vector radius (A.U.) %f",r);

omega=omega*pi/180;
i=i*pi/180;
w=w*pi/180;
e=e*pi/180;

PX=cos(omega)*cos(w)-cos(i)*sin(omega)*sin(w);
PY=sin(omega)*cos(w)*cos(e);
PY=PY+cos(i)*cos(omega)*sin(w)*cos(e);
PY=PY-sin(i)*sin(w)*sin(e);
PZ=sin(omega)*cos(w)*sin(e);
PZ=PZ+cos(i)*cos(omega)*sin(w)*sin(e);
PZ=PZ+sin(i)*sin(w)*cos(e);
QX=-cos(omega)*sin(w)-cos(i)*sin(omega)*cos(w);
QY=-sin(omega)*sin(w)*cos(e);
QY=QY+cos(i)*cos(w)*cos(omega)*cos(e);
QY=QY-sin(i)*cos(w)*sin(e);
QZ=-sin(omega)*sin(w)*sin(e);
QZ=QZ+cos(i)*cos(omega)*cos(w)*sin(e);
QZ=QZ+sin(i)*cos(w)*cos(e);

x=PX*r*cos(v*pi/180)+QX*r*sin(v*pi/180);
y=PY*r*cos(v*pi/180)+QY*r*sin(v*pi/180);
z=PZ*r*cos(v*pi/180)+QZ*r*sin(v*pi/180);

/*

Object's heliocentric ecuatorial rectangular coordinates.
Coordenadas rectangulares ecuatoriales helicéntricas del objeto

*/

printf("\n\n  Object's heliocentric ecuatorial rectangular coordinates");
printf("\n\n  x= %f y= %f z= %f", x, y, z);

GI=X+x;
ETA=Y+y;

```

```

ZETA=Z+z;

printf("\n\n Object's geocentric ecuatorial rectangular coordinates");
printf("\n\n GI= %f ETA= %f ZETA= %f", GI, ETA, ZETA);

alfa=atan(ETA/GI)*180/pi;
if (GI<0) alfa=alfa+180;
alfa=alfa*24/360;

hour = floor(alfa);
min = floor((alfa-hour)*60);
sec = floor((alfa - hour - min/60.0)*3600);

printf("\n Object's right ascension %d h, %d m, %d s", hour, min, sec);

delta=atan(ZETA/sqrt(GI*GI+ETA*ETA))*180/pi;

grado = floor(delta);
min = floor((delta-grado)*60);
sec = floor((delta - grado - min/60.0)*3600);

printf("\n Object's declination %d deg %d ', %d ''", grado, min, sec);

r=sqrt(GI*GI+ETA*ETA+ZETA*ZETA);

printf("\n Geocentric distance %f U.A.", r);

/* end */
}

```