Problemy implementacyjne przy tłumaczeniu z Fortrana na ObjectPascala

Przy pracy nad rozwiązaniem określającym pozycję Słońca, zwróciłem się do pana Kazimierza Borkowskiego[1] z Katedry Radioastronomii Uniwersytetu Mikołaja Kopernika w Toruniu, którego wiedza w tej dziedzinie dostarczała pewności, że obliczenia, jakie przeprowadzę przy użyciu jego rozwiązań, dadzą wymagane wyniki o wystarczającej dokładności. Kod programu w języku Fortran, świetny, choć dla niewtajemniczonego zupełnie niezrozumiały, skrypt[2] i inne opracowania zamieszczone na stronie internetowej tegoż autora, dawały szansę na szybkie wdrożenie tego pomysłu w życie. Potrzebowałem implementacji w języku, który umożliwia mi potem sterowanie urządzenia z komputera portem USB. Niestety okazało się, że zrozumienie składni języka Fortran to prawdziwa droga przez mękę, dla programisty obiektowego.

Początkowo nieufnie zabrałem się do tego zadania, gdyż wielu programistów współcześnie ma dość w sumie wypaczone zdanie, na temat tego szacownego języka, który w końcu jest pierwszym językiem wysokiego poziomu. Dopiero po bardzo długim czasie, ale na szczęście nie astronomicznym, udało mi się tej sztuki dokonać. W sumie nie ma czym się chwalić, jakby kto ujął to w nieco bardziej kolokwialnej formie, ale proszę samemu to zrealizować, a od tej chwili oddam głos gwiazdom. Nic bowiem w tej translacji nie było proste, w końcu ruch planet i gwiazd to złożony układ sumujących się harmonicznych, którego to sumowania nikt nie umie podać punktu początkowego.

Przyjąłem w końcu za punkt wyjścia algorytm Hatchera[3] podany[4] przez Kazimierza Borkowskiego, gdzie L, M i N lata, miesiące i dni daty kalendarza gregoriańskiego lub juliańskiego. W celu obliczenia dnia juliańskiego JD trzeba wykonać następujące rachunki:

$$L' = L + 4716 - \mathrm{int}[(14 - M)/12],$$
$$M' = (M + 9) \bmod 12,$$

dla kalendarza gregoriańskiego:

$$G = 0,$$

dla kalendarza juliańskiego:

$$G = \mathrm{int}\{3\,\mathrm{int}[(L' + 184)/100]/4\} - 38$$

i całość:

$$JD = \mathrm{int}(365{,}25L') + \mathrm{int}(30.6M' + 0.4) + N - G - 1402.$$

[1] http://www.astro.uni.torun.pl/~kb/kb.htm
[2] „Astronomiczne obliczenia nie tylko dla geografów”, Kazimierz Borkowski, Uniwersytet Mikołaja Kopernika, Toruń 1991.
[3] Hatcher D.A., 1985, *Q.J.R.A.Soc.* 26, 151.
[4] http://www.astro.uni.torun.pl/~kb/Artykuly/PA/Juldat.htm

Funkcja autorstwa Kazimierza Borkowskiego implementująca ten algorytm w Fortranie (Lahey), pozwalającym na automatyczne wykonanie funkcji `Int()`, wygląda tak:

```
FUNCTION JD(L,M,N,J1G0)

C   OBLICZA DZIEN JULIANSKI Z DATY KALENDARZA GREGORIANSKIEGO [J1G0=0]
C  LUB JULIANSKIEGO [J1G0=1]  (L = ROK, M = MIESIAC, N = DZIEN MIESIA-
C  CA). ALGORYTM DZIALA POPRAWNIE OD 1 MARCA -100100 R.  (OBU KALENDA-
C  RZY).

      JD=(L+(M-8)/6+100100)*1461/4+(153*MOD(M+9,12)+2)/5+N-34840408
      IF(J1G0.LE.0)          JD = JD-(L+100100+(M-8)/6)/100*3/4+752
END
```

Jako ciekawostka to samo w składni GNU Fortran kompilatora gfortran[5]:

```
function jd(l,m,n,j_g0) result (dj)
integer ::l,m,n,j_g0!deklaracja zmiennych nagłówkowych.
real :: dj!deklaracja zmiennej wyniku.

!Oblicza dzień juliański z daty kalendarza gregoriańskiego [j1g0=0]
!lub juliańskiego [j_g0=1] (l = rok, m = miesiąc, n = dzień miesiąca).
!Algorytm działa poprawnie od 1 marca -100100 r.  (obu kalendarzy).

      dj=(l+(m-8)/6+100100)*1461/4+(153*mod(m+9,12)+2)/5+n-34840408
      if(j_g0.le.0)          dj = dj-(l+100100+(m-8)/6)/100*3/4+752
end function jd
```

Oba te zapisy niestety onieśmielają. Ujmujące swą prostotą formalną i całkowicie nieintuicyjne dla niewtajemniczonego nie tylko w niuanse języka, ale w kwestie astronomiczne, zadają kłam temu, że język Fortran to niepotrzebny staroć, jak temu, że przetłumaczenie z niego kodu na inne *nowoczesne* języki to zadanie łatwe i proste.

Implementacja tej funkcji wprost z Fortrana nastręcza zatem niestety trudności, gdyż cechy języka ObjectPascal nie pozwalają na wykonanie domyślnie funkcji `Int()`. Dlatego trzeba wrócić do oryginalnego algorytmu Hatchera, co wygląda tak:

```
function JD(L,M,N,J1G0:Integer):Double;
//OBLICZA DZIEŃ JULIAŃSKI Z DATY KALENDARZA GREGORIAŃSKIEGO [J1G0=0]
//LUB JULIAŃSKIEGO [J1G0=1] (L = ROK, M = MIESIĄC, N = DZIEŃ MIESIĄCA.
var L_,M_,G:Double;
begin
   L_:=L+4716-Int((14-M)/12);
   M_:=(M+9) mod 12;
   if J1G0=0 then G:=0 else G:=Int(3*Int((L_+184)/100)/4)-38;
   JD:=Int(365.25*L_)+Int(30.6*M_+0.4)+N-G-1402;
end;
```

I na koniec to, czym dysponuje Delphi Borlanda, jako funkcją do obliczania dnia juliańskiego:

[5] Przy użyciu pakietu Edi 3,1 autorstwa Wojciecha Sobieskiego z Uniwersytetu Warmińsko-Mazurskiego.

```
{ Julian and Modified Julian Date conversion support }
function DateTimeToJulianDate(const AValue: TDateTime): Double;
var
   LYear, LMonth, LDay: Word;
begin
  DecodeDate(AValue, LYear, LMonth, LDay);
  Result := (1461 * (LYear + 4800 + (LMonth - 14) div 12)) div 4 +
            (367 * (LMonth - 2 - 12 * ((LMonth - 14) div 12))) div 12 -
            (3 * ((LYear + 4900 + (LMonth - 14) div 12) div 100)) div 4
            LDay - 32075.5 + Frac(AValue);
end;
```

Dla niewtajemniczonych powiem, że zmienna `TDataTime` jest zmiennoprzecinkową zakodowaną wartością czasu. Natomiast funkcja `DecodeDate()` odczytuje rok, miesiąc i dzień z tej wartości (`AValue`). Natomiast funkcja `Frac()` zwraca resztę po przecinku.

Nie wiem, co powiedzą astronomowie na temat tego ostatniego rozwiązania, w każdym razie, w programie do określania pozycji Słońca zrezygnowałem z zastosowania delphickiego rozwiązania, ufając jednak w wiedzę znakomitych astronomów, popartą korektami obserwacyjnymi. Znaczna różnica w wartości po przecinku, jaka powstaje między dniem juliańskim wyliczonym tą pozornie dość wygodną metodą wywołania funkcji bibliotecznej w stosunku do wyniku uzyskanego z funkcji `JD()`, zupełnie dyskredytuje rozwiązanie Delphi do dokładnych obliczeń. Pomijając już całkowicie milczeniem kwestię optymalizacji w rozwiązaniu firmowym autorów tego kompilatora.

Pozostałem zatem przy algorytmie Hatchera.

Reszta obliczeń nie mieści się w tym skromnym tekście, który miał na celu jedynie pokazać, że trudności w przekładzie między językami programistycznymi wynikają nie tylko z różnic syntaktycznych i składniowych, ale wprost pochodzą z innego podejścia do materii obliczeniowej u autorów tych języków. Niestety współcześni specjaliści zbyt często zapominają, że treścią języków programowania były i są obliczenia matematyczne, które wymagają wyrafinowanej wiedzy, a nie tylko łatwizny, jaką dostarcza powierzchowne często opracowanie producentów komercyjnie świetnie się prezentujących rozwiązań. To co ma wartość ponadczasową, to nie jest tendencją mody w programistyce, ale wiedzy dającej trwały nośnik wartości o nieco dłuższym średnim czasie życia niż ciasne myślenie merkantylizmu.

Andrzej Marek Hendzel