

case study

JULIAN DATES

SYNOPSIS

This case study looks at how the Gregorian to Julian conversion can be implemented in five programming languages: Fortran, Ada, Cobol, C and Python.

Type:	<i>comparative language study</i>
Language:	<i>Fortran, C, Python, Cobol, Ada</i>
Compiler:	<i>gfortran, gcc, python, cobc, gnatmake</i>
Skills:	<i>program design, language conversion</i>
Experience Level:	<i>novice</i>

*The die is cast.
Julius Caesar*

PROBLEM DESCRIPTION

Julian dates (abbreviated JD) are simply a continuous count of days and fractions since noon Universal Time on January 1, 4713 BCE (on the Julian calendar). Almost 2.5 million days have transpired since this date. Julian dates are widely used as time variables within astronomical software.

Fliegel and van Flandern (1968) published compact computer algorithms for converting between Julian dates and Gregorian calendar dates. Their algorithms were presented in Fortran, and take advantage of the truncation feature of integer arithmetic. In the code that follows, **year** is the full representation of the year, such as 1970, 2000, etc. (not a two-digit abbreviation); **month** is the month, a number from 1 to 12; **day** is the day of the month, a number in the range 1-31; and **jd** is the Julian date at Greenwich noon on the specified day, month, and year.

$$\begin{aligned} \text{JD}(\text{d}, \text{m}, \text{y}) = & \text{d} - 32075 + 1461 * (\text{y} + 4800 + (\text{m} - 14) / 12) / 4 \\ & + 367 * (\text{m} - 2 - (\text{m} - 14) / 12 * 12) / 12 - 3 \\ & * ((\text{y} + 4900 + (\text{m} - 14) / 12) / 100) / 4 \end{aligned}$$

This case study looks at how the Gregorian to Julian conversion can be implemented in five programming languages: Fortran, Ada, Cobol, C and Python. It provides a sense of how these programming languages differ from each other. Note that input to all programs is of the form:

DD/MM/YEAR

Each program will parse out the relevant data, ignoring the / symbols.

REF:

Fliegel, H.F., van Flandern, T.C. "A machine algorithm for processing calendar dates", *Communications of the ACM*, Vol.11(10), p.657 (1968)

FORTRAN 95

The first program for performing the conversion to Julian is written in Fortran 95.

```
! Program to calculate the Julian date given a Gregorian date
! in the format DD/MM/YYYY
! Derived from an algorithm by:
!   Fliegel, H.F., van Flandern, T.C., "A machine algorithm for processing
!   calendar dates", Communications of the ACM, Vol.11(10), pp.657 (1968)

program gregorian2julian

    integer :: day, month, year, dt

    ! Obtain the user input
    write (*,*) 'Date (DD/MM/YYYY): '
    read (*,50) day, month, year
    50 format (I2,1X,I2,1X,I4)

    dt = julian(day,month,year)

    ! Write the output
    write (*,100) dt
    100 format ('The Julian date is ', I10)

end program gregorian2julian

integer function julian(d, m, y)

    integer, intent(in) :: d, m, y

    ! Perform the conversion from Gregorian to Julian
    julian = d - 32075 + 1461 * (y + 4800 + (m - 14) / 12) / 4 &
        + 367 * (m - 2 - (m - 14) / 12 * 12) / 12 - 3 &
        * ((y + 4900 + (m - 14) / 12) / 100) / 4

end function julian
```

C

The second program for performing the conversion to Julian is written in C.

```
#include <stdio.h>

int julian(int d, int m, int y)
{
    int jd;

    jd = d - 32075 + 1461 * (y + 4800 + (m - 14) / 12) / 4
        + 367 * (m - 2 - (m - 14) / 12 * 12) / 12 - 3
        * ((y + 4900 + (m - 14) / 12) / 100) / 4;

    return jd;
}

int main(void)
{
    int day, month, year, dt;

    printf("Date (DD/MM/YYYY): ");
    scanf("%d/%d/%d", &day, &month, &year);

    dt = julian(day, month, year);

    printf("The Julian date is %d\n", dt);

    return 0;
}
```

ADA

The third program for performing the conversion to Julian is written in Ada.

```
with ada.text_io; use ada.text_io;
with ada.integer_text_io; use ada.integer_text_io;

procedure greg2julian is

    day, month, year, dt : integer;
    s1, s2 : character;

    function julian(d : in integer; m : in integer; y : in integer)
        return integer is
        jd : integer;
    begin
        jd := d - 32075 + 1461 * (y + 4800 + (m - 14) / 12) / 4
            + 367 * (m - 2 - (m - 14) / 12 * 12) / 12 - 3
            * ((y + 4900 + (m - 14) / 12) / 100) / 4;
        return jd;
    end julian;

begin

    put_line("Date (DD/MM/YYYY): ");
    get(day);
    get(s1);
    get(month);
    get(s2);
    get(year);

    dt := julian(day, month, year);

    put_line("The Julian date is " & integer'image(dt));

end greg2julian;
```

COBOL

The fourth program for performing the conversion to Julian is written in Cobol. Note here that Cobol does not do integer division, so it is necessary to use a function to truncate the decimal component.

```
identification division.  
program-id. greg2julian.
```

```
environment division.  
input-output section.  
file-control.
```

```
data division.  
file section.
```

```
working-storage section.
```

```
01  Gdate.  
    02  d           pic 99.  
    02  filler      pic x.  
    02  m           pic 99.  
    02  filler      pic x.  
    02  y           pic 9(4).  
  
77  jd             pic ZZZZ999999.  
77  temp1          pic S99.  
77  temp2          pic 9(8).  
77  temp3          pic S999.  
77  temp4          pic 9(8).
```

```
procedure division.
```

```
    display "Date (DD/MM/YYYY): ".  
    accept Gdate.
```

```
    perform julian.  
    display "The Julian date is ", jd.
```

```
stop run.
```

```
julian.
```

```
    compute temp1 = m - 14.  
    compute temp2 = d - 32075 + 1461 * (y + 4800 +  
        function integer-part(temp1 / 12)) / 4.  
    compute temp3 = function integer-part(temp1 / 12) * 12.  
    compute temp4 = ((y + 4900 + temp1 / 12) / 100).  
    compute jd = temp2 + 367 * (m - 2 - temp3) / 12 - 3 * temp4 / 4.
```

PYTHON

The fifth program for performing the conversion to Julian is written in Python. One of the caveats of Python is that it is dynamically typed, which means that care must be taken when operations such as integer division have to be performed.

```
import numpy
import math
from datetime import datetime

def julian(d,m,y):

    temp1 = m - 14
    temp2 = d - 32075 + 1461 * (y + 4800 + int(temp1 / 12.0)) / 4
    temp3 = int(temp1 / 12.0) * 12
    temp4 = ((y + 4900 + int(temp1 / 12.0)) / 100)
    jd = temp2 + 367 * (m - 2 - temp3) / 12 - 3 * temp4 / 4

    return jd

if __name__ == "__main__":

    date = raw_input("Date (dd/mm/yyyy): ")

    try:
        valid_date = datetime.strptime(date, '%d/%m/%Y')
    except ValueError:
        print('Invalid date!')

    day = valid_date.day
    month = valid_date.month
    year = valid_date.year

    jd = julian(day,month,year)

    print("The Julian date is %d \n" % (jd))
```

WATCH OUT FOR THE MOD

One thing to be careful for in some programming languages is the use of the **mod** function. A case in point is the piece of the equation:

$$(m - 2 - (m - 14) / 12 * 12)$$

It is tempting to re-write this portion of highlighted code, with the principle that:

$$r = x / y * y$$

is equivalent to:

$$r = x - \text{mod}(x, y)$$

Then write the code as:

```
temp = (m - 14)
r = temp - mod(temp/12)
```

This works fine as long as temp is a positive number. If it is negative, all bets are off.

$$\text{temp} = (m - 14)$$

This is because the modulus is normally calculated using the following equation:

$$\text{mod}(x, y) = x - (y * \text{INT}(x / y))$$

Where the function INT calculates the greatest integer less than or equal to the argument. Therefore:

$$\begin{aligned}\text{mod}(13, 12) &= 13 - (12 * \text{INT}(13 / 12)) \\ \text{mod}(13, 12) &= 1\end{aligned}$$

But, for negative numbers, a different answer is derived:

$$\begin{aligned}\text{mod}(-13, 12) &= -13 - (12 * \text{INT}(-13 / 12)) \\ \text{mod}(-13, 12) &= -13 - (12 * -2) \\ \text{mod}(-13, 12) &= 11\end{aligned}$$

So if there is no integer division (such as in Cobol), do this instead:

```
temp = (m - 14)
r = INT(temp / 12) * 12
```


INTEGER DIVISION CAN BE TRICKY TOO

In Python integer division is possible, but again negative numbers complicate things. Here it's basically because

$$r = x / y$$

is calculated using:

$$r = \text{floor}(x.0/y.0)$$

Therefore:

$$r = 8 / 7 = \text{floor}(8.0/7.0) = 1$$

while if one of the numerator or denominator is negative, a different answer evolves:

$$r = 8 / -7 = \text{floor}(8.0/-7.0) = \text{floor}(-1.143) = -2$$