The Calculation of Easter*

Ryan Stansifer
Department of Computer Sciences
University of North Texas
Denton, TX 76203–3886

November 3, 1994

In a 1962 article that appeared in the *Communications of the ACM*, Donald Knuth gave two programs for computing the date on which Easter falls. One was written in ALGOL, the other in COBOL. Besides having a fascinating history, this algorithm makes a good, small programming project. We offer here our programs written in Modula-3 and ML. Modula-3 is an imperative language with modules. Other distinctive features are objects, exception handling, garbage collection, and processes. The August 1992 issue of *SIGPLAN Notices* reprinted the reference manual for the language. ML is a strongly-typed functional language. Both languages have freely distributed implementations.

The Council of Nicaea in 325 A.D. decreed that Easter was to be the Sunday following the full moon which occurred on, or next after, the day of the spring equinox. This rule, however, is ambiguous, since the time and date of the full moon and the equinox depend on the point of observation. These problems were overcome by adopting the 21st of March as the equinox, and by assuming the full moons re-occur after an exact period of nineteen Julian years of $365\frac{1}{4}$ days. Over the course of a 1,000 years these approximations introduced an error in the equinox of about 10 days. This led to the reformation of the calendar promulgated by Pope Gregory XIII.

Knuth has refined and described the algorithm for calculating the date of Easter in the Gregorian calendar. We quote from Exercise 14 from Section 1.3.2 of *The Art of Computer Programming: Fundamental Algorithms*, page 155–156.

> The following algorithm, due to the Neapolitan astronomier Aloysius Lilius and the German Jesuit mathematician Christopher Clavious in the late 16th century, is used by most Western churches to determine the date of Easter Sunday for any year after 1582. [For previous years, see *CACM* **5** (1962), 209–210. The first systematic algorithm for calculating the date of Easter was the *canon paschalis* due to Victorius of Aquitania (457 A.D.). For further commentary, see *Puzzles and Paradoxes* by T. H. O'Beirne (London: Oxford University Press, 1965), Chapter 10.]
>
> **Algorithm E.** (*Date of Easter.*) Let Y be the year for which the date of Easter is desired.
>
> **E1.** [Golden number.] Set $G \leftarrow (Y \bmod 19) + 1$. ($G$ is the so-called "golden number" of the year in the 19-year Metonic cycle.)

*An earlier version of this paper appeared in *SIGPLAN Notices*, volume 27, number 12, December 1992, pages 61–65. In that version a `=` appeared in the programs instead of a `>`.

E2. [Century.] Set $C \leftarrow \lfloor Y/100 \rfloor + 1$. (When Y is not a multiple of 100, C is the century number; i.e., 1984 is in the twentieth century.)

E3. [Corrections.] Set $X \leftarrow \lfloor 3C/4 \rfloor - 12$, $Z \leftarrow \lfloor (8C+5)/25 \rfloor - 5$. ($X$ is the number of years, such as 1900, in which leap year was dropped in order to keep in step with the sun. Z is a special correction designed to synchronize Easter with the moon's orbit.)

E4. [Find Sunday.] Set $D \leftarrow \lfloor 5Y/4 \rfloor - X - 10$. [March $((-D) \bmod 7)$ actually will be a Sunday.]

E5. [Epact.] Set $E \leftarrow (11G+20+Z-X) \bmod 30$. If $E = 25$ and the golden number G is greater than 11, or if $E = 24$, then increase E by 1. (E is the so-called "epact," which specifies when a full moon occurs.)

E6. [Find full moon.] Set $N \leftarrow 44 - E$. If $N < 21$ then set $N \leftarrow N + 30$. (Easter is supposedly the "first Sunday following the first full moon which occurs on or after March 21." Actually perturbations in the moon's orbit do not make this strictly true, but we are concerned here with the "calendar moon" rather than the actual moon. The Nth of March is a calendar full moon.)

E7. [Advance to Sunday.] Set $N \leftarrow N + 7 - ((D + N) \bmod 7)$.

E8. [Get month.] If $N > 31$, the date $(N - 31)$`APRIL`; otherwise the date is N `March`.

The following Modula-3 program computes the day and month of Easter according to the Gregorian calendar using the algorithm above. Like Knuth's ALGOL and COBOL programs, it also computes Easter according to the Julian calendar. Unlike Knuth's programs we do not use the Gregorian calendar after a fixed year. Although the Gregorian calendar was first adopted in 1582, it was not adopted by England and the colonies in the new world until 1752. With this in mind we designed the Modula-3 program to compute Easter according to either method based on how the program is invoked.

This simple program illustrates just a few features of the Modula-3 language. The program uses some of the standard, system-provided modules. The `IMPORT` construct makes a module's capabilities accessible. The `Params` module provides access to the command line with which the program is invoked. The integer `Params.Count` is the number of words on the line. The function `Params.Get` returns the word as an Modula-3 string (`Text.T`) given the index. The program must be invoked with one argument, the year. If not, then the program asks the run-time system to halt execution with a non-zero return status (traditionally used to indicate an unsuccessful conclusion). Some run-time facilities are provided by the `RTMisc` module.

The `Easter` program does some simple output. Writing requires the module `Wr`; directing the output to the terminal requires the device `stdout` from the module `Stdio`. The function `Wr.PutText` writes a single string, but concatenating several strings together is easy using the binary, infix operator `&`. Converting other types to strings is done by the `Fmt` module. In particular the function `Fmt.Int` converts an integer to a string. Converting the other way, from a string to an integer, requires the function `Scan.Int` from the `Scan` module.

The program `Easter` shows the use of the `WITH` statement for the introduction of constant values into some scope. "Constant" here is used in the sense of values not changed by the programmer. Modula-3 constant declarations are not possible in this case as constants must be values known to the compiler at compile time. This use of the `WITH` statement is much like the `let` construct in functional languages. The `WITH` statement is also used in Modula-3 to introduce aliases. In this repect the statement is like the `WITH` statement in Pascal.

Like Pascal, Modula-3 has call-by-value and call-by-reference parameter-passing modes. The program `Easter` illustrates Modula-3's third mode: `READONLY`. The `READONLY` formal parameters are used to make explicit that the parameters are not changed by the programmer. Any attempt to assign to a `READONLY` parameter is flagged as an error by the compiler. Hence, either method of parameter passing could be used, and the implementation may pick the most efficient one.

```
MODULE Easter EXPORTS Main;  (* Program to compute date of Easter *)

IMPORT Params, RTMisc;
IMPORT Fmt, Scan, Stdio, Wr, Text;

PROCEDURE Param1 (): INTEGER =
    BEGIN
        IF Params.Count > 1
        THEN
            RETURN (Scan.Int (Params.Get(1)));
        ELSE
            Wr.PutText (Stdio.stdout, "Year required as argument\n");
            RTMisc.Exit (1);
        END
    END Param1;


PROCEDURE F (READONLY E, G: INTEGER): INTEGER =
    BEGIN
        IF E=24 OR (E=25 AND G>11)
        THEN
            RETURN (E+1)
        ELSE
            RETURN (E);
        END
    END F;


PROCEDURE G (READONLY N: INTEGER): INTEGER =
    BEGIN
        IF N<21
        THEN
            RETURN (N+30)
        ELSE
            RETURN (N);
        END
    END G;

PROCEDURE EM (READONLY N: INTEGER): Text.T =   (* Easter month      *)
    BEGIN
        IF N>31
        THEN
            RETURN ("April")
        ELSE
            RETURN ("March");
        END
    END EM;
```

```
PROCEDURE ED (READONLY N: INTEGER): INTEGER =  (* Easter date       *)
    BEGIN
        IF N>31
        THEN
            RETURN (N-31)
        ELSE
            RETURN (N);
        END
    END ED;

VAR
    Year   := Param1 ();          (* Get year from command line.    *)
    Julian := Params.Count > 2;   (* If 3rd arg, then Julian calen. *)

    Golden  := (Year MOD 19) + 1;

    Extra     : INTEGER;   (* Where Sunday falls in the year.       *)
    Epact     : INTEGER;   (* Age of calendar moon Jan 1.           *)
    Moon      : INTEGER;   (* Full moon after equinox.              *)
    Sunday    : INTEGER;   (* Easter Sunday in days after March 1.  *)

BEGIN
    IF Julian
    THEN
        Extra := (5*Year DIV 4);
        Epact := (11*Golden - 4) MOD 30 + 1;
    ELSE
        WITH
           Century   = (Year DIV 100) + 1,
           Gregorian = (3*Century DIV 4) - 12,
           Clavian   = ((8*Century + 5) DIV 25) - 5
        DO
           Extra := (5*Year DIV 4) - Gregorian - 10;
           Epact := F ((11*Golden+20+Clavian-Gregorian) MOD 30, Golden);
        END;
    END;

    Moon      := G (44 - Epact);
    Sunday    := Moon + 7 - ((Extra+Moon) MOD 7);

    Wr.PutText (Stdio.stdout, "Year = "      & Fmt.Int (Year) & "\n");
    Wr.PutText (Stdio.stdout, "EasterMon = " & EM (Sunday) & "\n");
    Wr.PutText (Stdio.stdout, "EasterDay = " & Fmt.Int (ED (Sunday)) & "\n");
END Easter.
```

The Modula family of programming languages tends to be verbose. For instance, subprogram definitions and `IF` statements take up many lines of code. In more substantial programs this overhead is less noticeable and syntax may lead to a more readable program. The approximately 90 lines taken to write the program in Modula-3 is very nearly the same number of lines that the COBOL program takes. (It must be mentioned that Knuth's COBOL program actually does more work by printing formatted pages of Easter dates from 500 to 4999 A.D.) The Modula-3 program, however, is much clearer. Knuth's ALGOL program is quite concise. One reason for this is the conditional expression—something that Modula-3 lacks.

For the sake of comparison we give an ML function that computes the date of Easter as well. In the function `easter`, the computation of the epact using the Gregorian method is factored out into its own function `G`. Like the subprocedures in the Modula-3 program, `G` is a local, auxiliary function to the main body of computation. This is accomplished in ML by the `local-in-end` construct. The `local` construct should not be confused with the `let` construct. In the `let` construct an expression, not a declaration, comes between the `in` and the `end`.

Pairs of values also play a role in the ML program. The function `G` returns the pair of integers `Extra` and `Epact`. The function `easter` returns the month and day of Easter, a string and integer pair.

```
local
    fun G (year, golden) =
        let
           val Century   = (year div 100) + 1;
           val Gregorian = (3*Century div 4) - 12;
           val Clavian   = ((8*Century + 5) div 25) - 5;
           val Extra     = (5*year div 4) - Gregorian - 10;
           val y = (11*golden+20+Clavian-Gregorian) mod 30; (* 0 <= y < 30 *)
        in
           (Extra, if y=24 orelse (y=25 andalso golden>11) then y+1 else y)
        end;
in
    fun easter (year, julian) =
        let
            val Golden = (year mod 19) + 1;
            val (Extra, Epact) =
                if julian
                    then ((5*year div 4), (11*Golden - 4) mod 30 + 1)
                    else G (year, Golden);
            val Moon = let val x = 44-Epact in if x<21 then x+30 else x end;
            val Sunday = Moon + 7 - ((Extra+Moon) mod 7);
        in
            if Sunday>31 then ("April", Sunday-31) else ("March", Sunday)
        end;
end;
```

1993	April	11	2014	April	20	2035	March	25	2056	April	2	2077	April	11
1994	April	3	2015	April	5	2036	April	13	2057	April	22	2078	April	3
1995	April	16	2016	March	27	2037	April	5	2058	April	14	2079	April	23
1996	April	7	2017	April	16	2038	April	25	2059	March	30	2080	April	7
1997	March	30	2018	April	1	2039	April	10	2060	April	18	2081	March	30
1998	April	12	2019	April	21	2040	April	1	2061	April	10	2082	April	19
1999	April	4	2020	April	12	2041	April	21	2062	March	26	2083	April	4
2000	April	23	2021	April	4	2042	April	6	2063	April	15	2084	March	26
2001	April	15	2022	April	17	2043	March	29	2064	April	6	2085	April	15
2002	March	31	2023	April	9	2044	April	17	2065	March	29	2086	March	31
2003	April	20	2024	March	31	2045	April	9	2066	April	11	2087	April	20
2004	April	11	2025	April	20	2046	March	25	2067	April	3	2088	April	11
2005	March	27	2026	April	5	2047	April	14	2068	April	22	2089	April	3
2006	April	16	2027	March	28	2048	April	5	2069	April	14	2090	April	16
2007	April	8	2028	April	16	2049	April	25	2070	March	30	2091	April	8
2008	March	23	2029	April	1	2050	April	10	2071	April	19	2092	March	30
2009	April	12	2030	April	21	2051	April	2	2072	April	10	2093	April	12
2010	April	4	2031	April	13	2052	April	21	2073	March	26	2094	April	4
2011	April	24	2032	March	28	2053	April	6	2074	April	15	2095	April	24
2012	April	8	2033	April	17	2054	March	29	2075	April	7	2096	April	15
2013	March	31	2034	April	9	2055	April	18	2076	April	19	2097	March	31