GAUSS' CALENDAR FORMULA FOR THE DAY OF THE WEEK

BERNDT E. SCHWERDTFEGER

In memory of my father Martin Walter (1910–1977)

ABSTRACT. This paper describes a formula of C. F. GAUSS for calculating the day of the week from the calendar date and gives a conceptual proof. A formula for the *Julian day number* is derived. My implementation of the formula in REXX and C is included.

PREFACE

GAUSS' formula calculates the day of the week, *Monday*, ..., *Sunday*, from the calendar date `year-month-day`. I learned about it in 1962 when reading the *Mathematical Recreations* [2] by MAURICE KRAITCHIK (1882–1957). It is stated there without proof, together with an application to a perpetual calendar.

This article explains the formula and its mathematical background, including a proof and the concepts behind the algorithms implemented in the program `gauss.c`.

GAUSS never published his formula; it appeared only in 1927 in his *Werke* [1, [XI, 1.]. For the convenience of the reader I have included this notice in appendix A.1

Berlin, 11 May 2009

 v1.4a, 14 November 2018

1. GAUSS' FORMULA EXPLAINED

1.1. **The Gauss formula.** Given are d = day, m = month, y = year. We split the year into the century part and the two digit year inside the century. For $m \geq 3$ it is done for the year $y = 100 \cdot c + g$, i.e. $c = [y/100]$, $g = y - 100 \cdot c$, such that $0 \leq g \leq 99$. For $m = 1, 2$ it is done for the year $y - 1 = 100 \cdot c + g$.

Following KRAITCHIK [2, The Calendar, p. 110–111] GAUSS' formula is:

$$w \equiv d + e + f + g + [g/4] \pmod 7$$

where $w \equiv 1, \ldots$ corresponds to *Monday*, ..., etc., and with the following constants depending on the month (e) resp. century (f):

Month	m	1	2	3	4	5	6	7	8	9	10	11	12
	e	0	3	2	5	0	3	5	1	4	6	2	4

Gregorian Calendar	Century	$c \bmod 4$	f
	1600, 2000, ...	0	0
	1700, 2100, ...	1	5
	1800, 2200, ...	2	3
	1900, 2300, ...	3	1

2010 *Mathematics Subject Classification.* Primary 68-04; Secondary 68N15.
Key words and phrases. Calendar formula, perpetual calendar, Julian day number.

For the *Julian calendar* the same formula applies with modified constants f according to the following table:

Julian Calendar	Century	$c \bmod 7$	f
	$0000, 0700, 1400, \ldots$	0	5
	$0100, 0800, 1500, \ldots$	1	4
	$0200, 0900, 1600, \ldots$	2	3
	$0300, 1000, 1700, \ldots$	3	2
	$0400, 1100, 1800, \ldots$	4	1
	$0500, 1200, 1900, \ldots$	5	0
	$0600, 1300, 2000, \ldots$	6	6

Some *perpetual calendars* exploit this formula (or present tables that are equivalent to the formula). A short explanation (with some gaps) of this formula is given in the next section 1.2 and a complete proof in section 2.

1.2. **A short reasoning.** First, recall the Gregorian rule for *leap years*:

A normal year has 365 *days, with an extra day – the leap day 29.02. – added every four years, except for century years, when the century is not divisible by* 4.

In terms of our variables this reads

$$y = 100 \cdot c + g \quad \text{is a leap year} :\Longleftrightarrow$$
$$(g \neq 0 \text{ and } g \equiv 0 \bmod 4) \text{ or } (g = 0 \text{ and } c \equiv 0 \bmod 4)$$

This rule implements a periodicity of 400 years that have exactly $3 \cdot 24 + 25 = 97$ leap years and $3 \cdot 76 + 75 = 303$ normal years. Otherwise put, 400 consecutive years in the Gregorian Calendar consists always of exactly $97 \cdot 366 + 303 \cdot 365 = 146097$ days, which results in an average length of a Gregorian year of $\frac{146097}{400} = 365.2425$ days (or, if you prefer, has 31,556,952 seconds).

A brief 'reasoning' on this formula (with some shortcuts) runs as follows:

The first term d, the progression with days, is obvious.

$365 \equiv 1 \pmod 7$, so the first and last day of any normal year fall on the same weekday. This explains the term g (from year to year progress). The other term $[g/4]$ accounts for the extra day every four years.

How much week days do we have to shift from one century to the next? $100 \equiv 2 \pmod 7$, plus the 24 leap years in the century give $24 \equiv 3 \pmod 7$, so a century adds $2+3 = 5$ shift days.

For the Julian Calendar we have 25 leap years in a century, thus a century shifts by $2 + 4 = 6$ days.

This explains the f values: $5 + 5 \equiv 3 \pmod 7$, $5 + 5 + 5 \equiv 1 \pmod 7$. Their dependency on $c \pmod 4$ and the gap ('leap') from $c \equiv 3$ to $c \equiv 0 \pmod 4$ is due to the leap century rule (in the Gregorian Calendar).

This leaves us to explain the 'magic' values for e. Let us look at the following table with the number $d(m)$ of days in a month, the sum $e(m)$ of days for the previous months (number of days from begin of the year for non leap years only) and their values mod 7:

m	1	2	3	4	5	6	7	8	9	10	11	12
$d(m)$	31	28	31	30	31	30	31	31	30	31	30	31
$e(m)$	0	31	59	90	120	151	181	212	243	273	304	334
(mod 7)	0	3	3	6	1	4	6	2	5	0	3	5

The 'magic' value is $e = e(m)$ for the months $m = 1, 2$ and $e = e(m) - 1$ for the other months $\ldots \pmod 7$.

It is left to the reader as an exercise to render this reasoning into an irreproachable mathematical proof and to develop a more conceptual approach.

Instead, you might as well study the program in the appendix B.2: the function `offset` in `gauss.c` gives a different parametrization of dates in terms of *offset* and *year* ; the function `date` is inverse to `offset`.

Or, you might want to read the conceptual approach developed in section 2.

1.3. **Historical Perspective.** The actual internationally used calendar is derived from the Roman calendar as reformed by Julius Caesar in 46 BC, who introduced the leap day every four years (*Julian Calendar*). The average length of the Julian year is therefore 365.25 days.

Now, the *solar* (or *tropical*) year, which is founded on the revolution of the Earth around the Sun, defined by passing the *vernal equinox*, is measurably shorter: 365 days, 5 hours, 48 minutes and 45.51 seconds, or 31,556,925.51 s, or 365.242193402 days. This is about 11 minutes and 15 seconds shorter than the Julian average year.

In the course of the centuries this discrepancy added up, leading to the fact that the Julian year was behind schedule: it had inserted too many leap days. In the 16^{th} century the deviation amounted to about 12 days. UGO BONCOMPAGNI (Bologna 1502 – Roma 1585), better known as Pope GREGORY XIII (1572), established in 1582 a calendar reform (*Gregorian Calendar*), which dropped 10 days: the day following (Julian) October 4 became (Gregorian) October 15, and introduced the leap year rules described above.

Note, however, the Gregorian year is not perfect for all times. It is (again) too long: but this time by only 26.49 seconds, but this will also add up over time. The imperfection accumulates to a whole day in about $1/(365.2425 - 365.242193402) = 3261$ years. Then, an additional leap day would have to be left out again ... With the advent of leap *seconds* this will, of course, not happen: clocks nowadays tick much more accurate than heavenly bodies once did (they didn't – but we didn't know).

As the scale for numbering the years is changing occasionally in our History, we will have to make a choice. In this paper we will treat all Calendar dates as if the Gregorian Calendar is valid thru eternity (so called *proleptic* Gregorian Calendar). We will also make use of the astronomical numbering of years by integers $\ldots, -3, -2, -1, 0, 1, 2, \ldots$ (instead of the Historians' numbering of 4 BC, 3 BC, 2 BC, 1 BC, 1 AD, 2 AD). Ignoring the possibly finite number of years since Creation, we will use the rational integers for the years.

The program `gauss.c` ignores the historical truth as well and neglects the Julian rules in force from 46 BC up to 1582 (and beyond in several countries). Instead, it treats all input dates according to Gregorian rules, but it gives the date of the Julian Calendar in its output.

2. TOWARDS A CONCEPTUAL APPROACH

2.1. **Heuristical remarks.** In principle the days from now to the past or to the future look like the set $\mathbf{Z}$ of rational integers. But there is no natural *day zero* and one has to make an arbitrary choice. We will see one way to do this numbering in section 2.4.

The set T of dates is another funny renumbering of $\mathbf{Z}$ by a triple $t = (y, m, d)$ of year, month, day (in some countries noted as `d.m.y`, in others `d/m/y` or even `m/d/y`; the ISO 8601 format prescribes `y-m-d` and this is also used in my program `gauss.c`.

The searched for map from T to the set of weekdays {Monday,..., Sunday} should be as simple as the canonical map $\mathbf{Z} \to \mathbf{F}_7$, the integers (mod 7). We will construct a candidate, to be called *Gauss maps*

$$\omega : T \longrightarrow \mathbf{F}_7$$

in a fairly natural way and give an explicit formula in terms of the coordinates y, m, d — it turns out to be the *Formula of Gauss* (8) in Corollary 2.3.

2.2. **The leap function.** For a year $y \in \mathbf{Z}$ we set the *century* $c = [y/100]$ and the remainder $g = y - 100 \cdot c$, such that $0 \le g \le 99$. The (Gregorian) leap function on the set of years is a boolean map

$$\lambda : \mathbf{Z} \longrightarrow \{0, 1\}$$

given by

$$\lambda(y) = \begin{cases} 1 & \text{if} \quad (g \neq 0 \wedge g \equiv 0 \bmod 4) \vee (g = 0 \wedge c \equiv 0 \bmod 4), \\ 0 & \text{otherwise} \end{cases}$$

The subset of leap years will be denoted by

$$\Lambda = \{y \in \mathbf{Z} \mid \lambda(y) = 1\}$$

We have

$$\Lambda = 400\mathbf{Z} \cup \bigcup_{k \neq 0 \bmod 25} 4k + 100\mathbf{Z}$$

We also define for a month m and any year y

$$\lambda(y, m) = \begin{cases} \lambda(y) & \text{if } m > 2 \\ 0 & \text{for } m = 1, 2 \end{cases}$$

As we need to keep track of counting leap years thru the centuries, let us note some properties of λ, more or less obvious from the definitions.

Lemma 2.1. *Let* $0 \le g < 100$. *We have*

$$\sum_{l=1}^{c} \lambda(100 \cdot l) = [c/4] \qquad c \ge 0 \tag{1}$$

$$\sum_{k=1}^{g} \lambda(100 \cdot c + k) = [g/4] \tag{2}$$

$$\sum_{x=1}^{100c+g} \lambda(x) = [c/4] + 24 \cdot c + [g/4] \qquad c \ge 0 \tag{3}$$

$$-\sum_{x=1+100c+g}^{0} \lambda(x) = [c/4] + 24 \cdot c + [g/4] \qquad c < 0 \tag{4}$$

Proof. The relations (1), (2) are obvious by the leap year rule. As to (3):

$$\begin{aligned} \sum_{y=0}^{100c+g} \lambda(y) &= \sum_{l=0}^{c-1} \sum_{k=0}^{99} \lambda(100 \cdot l + k) + \sum_{k=0}^{g} \lambda(100 \cdot c + k) \\ &= \sum_{l=0}^{c} \lambda(100 \cdot l) + \sum_{l=0}^{c-1} \sum_{k=1}^{99} \lambda(100 \cdot l + k) + \sum_{k=1}^{g} \lambda(100 \cdot c + k) \\ &= [c/4] + 1 + \sum_{l=0}^{c-1} 24 + [g/4] \\ &= [c/4] + 1 + 24 \cdot c + [g/4] \qquad \text{and } \lambda(0) = 1. \end{aligned}$$

As to (4): by symmetry $\lambda(-x)=\lambda(x)$, hence

$$\sum_{x=1+100c+g}^{0}\lambda(x)=\sum_{x=0}^{-1-100c-g}\lambda(x)$$

and we are in the situation of (3) with $-1-100c-g=100(-1-c)+(100-1-g)$, $-1-c\geq 0$, $0\leq 100-1-g<100$,

$$\begin{aligned}\sum_{x=1+100c+g}^{0}\lambda(x)&=1+\sum_{x=1}^{-1-100c-g}\lambda(x)=\\&=1+\left[\frac{-1-c}{4}\right]+24(-c-1)+\left[\frac{100-1-g}{4}\right]=\\&=1+\left[\frac{-1-c}{4}\right]-24\cdot c-24+25+\left[\frac{-1-g}{4}\right]=\\&=-[c/4]-24\cdot c-[g/4]\end{aligned}$$

where we have used the relation $1+\left[\frac{-1-n}{4}\right]=-\left[\frac{n}{4}\right]$, valid for any $n\in\mathbf{Z}$. □

2.3. **Shifting Dates.** The set T of valid 'dates' is a subset of

$$\mathbf{Z}\times\{1,\dots,12\}\times\{1,\dots,31\}$$

defined by

$$\begin{aligned}T:=&\,\mathbf{Z}\times\{1,3,5,7,8,10,12\}\times\{1,\dots,31\}\\&\cup\mathbf{Z}\times\{4,6,9,11\}\times\{1,\dots,30\}\\&\cup\mathbf{Z}\times\{2\}\times\{1,\dots,28\}\\&\cup\Lambda\times\{2\}\times\{29\}\end{aligned}$$

The last set in the union are the leap days (29^{th} of February).

We have a finite 'fibration' map

$$\pi:T\longrightarrow\mathbf{Z}$$

given by the first projection $\pi(y,m,d)=y$. Normal fibers $\pi^{-1}(y)$, $y\notin\Lambda$, have 365 elements, whereas 'leap' fibers ($y\in\Lambda$) have 366, i.e. fiber cardinality is

$$|\pi^{-1}(y)|=365+\lambda(y)\tag{5}$$

We use the natural order of days: $t<t'$ if t' is later than t. The cardinality of the segment of the year of all days up to $t=(y,m,d)$ is

$$|\{t'\in\pi^{-1}(y)\mid t'\leq t\}|=d+e(m)+\lambda(y,m)\tag{6}$$

We have a natural *shift* operation of the additive group $\mathbf{Z}$ on T:

$$\begin{aligned}\sigma:\mathbf{Z}\times T&\longrightarrow T\\(n,t)&\longmapsto n+t\end{aligned}$$

where $n+t$ is defined as t shifted by n days, earlier ($n<0$) or later ($n>0$) in time. We also write $\sigma_t(n)=\sigma(n,t)=n+t$, as well as $t'-t=n$ for $t'=n+t$.

Examples

$$\begin{aligned}1+(2000,2,28)&=(2000,2,29)\\1+(2001,2,28)&=(2001,3,1)\\-365+(2001,1,1)&=(2000,1,2)\\-152930+(2001,6,30)&=(1582,10,15)\\1872+(1947,2,4)&=(1952,3,21)\\(y,m,d)&=d+e(m)+\lambda(y,m)+(y-1,12,31)\end{aligned}$$

This operation is simply transitive. Therefore, any $t \in T$ gives rise to an isomorphism

$$\begin{aligned} \sigma_t : \mathbf{Z} &\xrightarrow{\sim} T \\ n &\longmapsto n + t \end{aligned}$$

which is *linear*: $\sigma_t(n+m) = n + \sigma_t(m)$. Taking the inverse σ_t^{-1} and combining with $\mathbf{Z} \to \mathbf{F}_7$ gives us *Gauss* maps

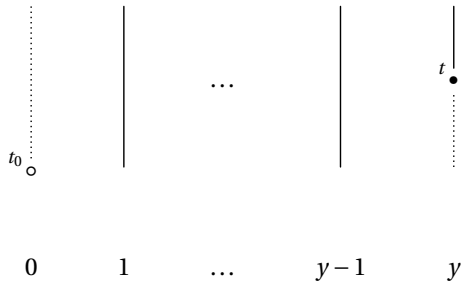
$$\omega_t : T \longrightarrow \mathbf{F}_7$$

which have the property $\omega_t(n+u) = n + \omega_t(u) = \omega_{-n+t}(u)$. So, there are only 7 of them, differing from each other by a constant $\in \mathbf{F}_7$.

Now I choose $t_0 = (0, 12, 31)$ as the arbitrary origin of our counting.

Let $t = (y, m, d) \in T$ be any date. To calculate $\omega(t) = \omega_{t_0}(t)$ we need to determine $n \in \mathbf{Z}$ with $n = t - t_0$. Then $\omega(t) = n \bmod 7 \in \mathbf{F}_7$.

To this end, we will have to go thru the fibers from t to t_0, from the fiber $\pi^{-1}(y)$ to the fiber $\pi^{-1}(1)$, and keep track of the number of days in passing.



(in this picture we assume $y > 0$).

Proposition 2.2.

$$t - t_0 = d + e(m) + 365 \cdot (y-1) + [c/4] + 24 \cdot c + [g/4] \tag{7}$$

Proof. Let us first assume $y > 0$, then by (6) and (5)

$$\begin{aligned} t - t_0 &= d + e(m) + \lambda(y,m) + \sum_{x=1}^{y-1} |\pi^{-1}(x)| = \\ &= d + e(m) + \lambda(y,m) + \sum_{x=1}^{y-1} (365 + \lambda(x)) = \\ &= d + e(m) + 365 \cdot (y-1) + \begin{cases} \sum_{x=1}^{y} \lambda(x) & m \geq 3 \\ \sum_{x=1}^{y-1} \lambda(x) & m = 1,2 \end{cases} \\ &= d + e(m) + 365 \cdot (y-1) + \sum_{x=1}^{100c+g} \lambda(x) = \\ &= d + e(m) + 365 \cdot (y-1) + [c/4] + 24 \cdot c + [g/4] \end{aligned}$$

where we have substituted $y = 100 \cdot c + g$ resp. $y - 1 = 100 \cdot c + g$, and applied the Lemma in section 2.2, relation (3).

Let us now assume $y \leq 0$, we are in the situation of the following picture:

$$y-1 \qquad y \qquad \dots \qquad -1 \qquad 0$$

and counting similarly the fibers we get

$$\begin{aligned} t - t_0 &= d + e(m) + \lambda(y,m) - \sum_{x=y}^{0} |\pi^{-1}(x)| = \\ &= d + e(m) + \lambda(y,m) - \sum_{x=y}^{0} (365 + \lambda(x)) = \\ &= d + e(m) + 365\cdot(y-1) - \sum_{x=1+100c+g}^{0} \lambda(x) = \\ &= d + e(m) + 365\cdot(y-1) + [c/4] + 24\cdot c + [g/4] \end{aligned}$$

where we have substituted $y = 100\cdot c + g$ resp. $y - 1 = 100\cdot c + g$, and applied the Lemma in section 2.2, relation (4). □

Corollary 2.3. *(Gauss formula)*

$$\omega(y,m,d) = d + e + f + g + [g/4] \tag{8}$$

where $f = 5\cdot r$ *when* $c = q\cdot 4 + r$, $0 \leq r \leq 3$ *and* $e = \begin{cases} e(m) & \text{for } m = 1,2 \\ e(m) - 1 & \text{for } m > 2 \end{cases}$

Proof. We have $\omega(t) = t - t_0 \mod 7$, hence by the proposition

$$\omega(t) = d + e(m) + y - 1 + [c/4] + 3\cdot c + [g/4]$$

For $m = 1,2$, as well as for $m > 2$

$$e(m) + y - 1 = e(m) - 1 + y = e + 100\cdot c + g$$

therefore

$$\begin{aligned} \omega(t) &= d + e + 5\cdot c + [c/4] + g + [g/4] = \\ &= d + e + 5\cdot(q\cdot 4 + r) + q + g + [g/4] = \\ &= d + e + f + g + [g/4] \end{aligned}$$

□

2.4. **The Julian day number.** The *Julian day number* is defined as the number of days passed since `JC -4712-01-01`, which is `-4713-11-24` in the proleptic Gregorian Calendar, hence is given by $j(t) = t - t_1 = (t - t_0) - (t_1 - t_0)$, where $t_1 = (-4713, 11, 24)$. From equation (7) we evaluate

$$t_1 - t_0 = 24 + e(11) + 365\cdot(-4713 - 1) + [-48/4] + 24\cdot(-48) + [87/4] = -1721425$$

hence

$$j(t) = 1721060 + d + e(m) + 365\cdot y + [c/4] + 24\cdot c + [g/4] \tag{9}$$

REFERENCES

[1] Carl Friedrich Gauss, *Werke* (1863), available at https://gdz.sub.uni-goettingen.de/id/PPN235957348.
[2] Maurice Kraitchik, *Mathematical Recreations*, George Allen & Unwill Ltd., London, 1955.

APPENDIX A. THE FORMULA IN GAUSS' NACHLASS

GAUSS never published his formula for finding the day of the week. It appeared in 1927 in [1, XI, 1.] p. 206 with remarks by ALFRED LOEWY regarding its derivation.

A.1. **Den Wochentag des 1. Januar eines Jahres zu finden.**

Handschriftliche Eintragung in: Sammlung astronomischer Tafeln, unter Aufsicht der Kgl. Preußischen Akademie der Wissenschaften, I. Band, Berlin 1776. — 1798 von GAUSS erworben.

Bezeichnet man den kleinsten positiven Rest einer Grösse A nach dem Modulus m durch $R:A \pmod m$, so lassen sich alle Vorschriften des Gregorianischen Kalenders auf folgende geschmeidige Art darstellen:

1.

Wenn man die Tage vom 1^{ten} Januar 1701 an zählt, d. i. diesen mit 1, den 2^{ten} mit 2, den 31^{ten} Dec. 1700 mit 0, den 30^{ten} mit -1 etc. bezeichnet, so ist der 1^{te} Januar in irgend einem Jahre A

$$\begin{aligned} = 1 + (A-1701)365 &+ \frac{1}{4}\big((A-1701) - R:(A-1701) \bmod 4\big) \\ &- \frac{1}{100}\big((A-1701) - R:(A-1701) \bmod 100\big) \\ &+ \frac{1}{400}\big((A-1601) - R:(A-1601) \bmod 400\big) \end{aligned}$$

2.

Die Wochentage Sonntag, Montag, etc. mit 0, 1 etc. bezeichnet, ist der 1^{te} Januar irgend eines Jahres, qua Wochentag,

$$\left.\begin{aligned} \equiv 6 + A &+ (2A + 5R:(A-1) \bmod 4) \\ &+ (3A + 4R:(A-1) \bmod 100) \\ &+ (A + 2 + 6R:(A-1) \bmod 400) \end{aligned}\right\} \bmod 7.$$

Also von 1601 bis 2000

$$\equiv 6 + 6A + 5R:(A-1) \bmod 4 + 4R:(A-1) \bmod 100.$$

Allgemein

$$\equiv 1 + 5R:(A-1) \bmod 4 + 4R:(A-1) \bmod 100 + 6R:(A-1) \bmod 400.$$

A.2. **Comparison.** The reader may look up the remarks by LOEWY in [1]. I contend that the formula in A.1 is identical to (8). Using earlier notation, let $y = A - 1 = 100c + g$ and $c = 4q + r$, then in GAUSS' notation $R:y \bmod m = y - [y/m]m$, hence

$$\begin{aligned} R:y \bmod 4 &= R:g \bmod 4 = g - 4[g/4] \\ R:y \bmod 100 &= g \\ R:y \bmod 400 &= 100r + g \end{aligned}$$

Calculating in $\mathbf{F}_7$ the weekday of January 1^{st} of year A by A.1 is

$$\begin{aligned} &= 1 + 5R:y \bmod 4 + 4R:y \bmod 100 + 6R:y \bmod 400 = \\ &= 1 + 5g - 20[g/4] + 4g + 12r + 6g = \\ &= 1 + g + [g/4] + 5r = \omega(A,1,1) \qquad \text{by formula (8).} \quad \square \end{aligned}$$

APPENDIX B. PROGRAM LISTINGS

The program `gauss` displays

- the day of the week ({Mon, Tue, Wed, Thu, Fri, Sat, Sun})
- the Gregorian date,
- the Julian date (`JC`),
- the day of the year (`D#`),
- the week of the year (`W#`),
- the Julian day number (`J#`),
- the Unix day (`X#`)

The input syntax is `date [offset]`, where *date* is in ISO 8601 format `year-mm-dd` and *offset* can be any integer. Weeks run from Monday to Sunday, its numbering follows ISO.

When experimenting with some input you will see that `gauss` is tolerant to some false dates (correcting them): asking for `1999-2-29` it correctly answers `1999-03-01`. Empty input will terminate the program.

B.1. **Earlier implementations.**

The subroutines go back to an implementation in REXX in the late 1980s on the IBM VM/CMS operating system. They were used in a program to generate weekly statistics of defects in a project. Then, at the wake of another leap year (1996), I put them all together into a small REXX program (`gauss.rexx`, see below in section B.3 for a recent version) and wrote up a *rationale* of the algorithm and a proof of the formula. This rough explanation of *Gauss'* formula constituted the original content at my Web site (June 1999).

In March 2004 I ported the program to Perl, and in March 2009 to C. Since 2004 the *Julian day number* and the Julian Calendar date was included.

The subroutine `date` implements the operation of the integers on the dates and `offset` returns the day of the year of a date. The subroutine `leap` implements the Gregorian leap year rules and `weekday` returns the result of *Gauss'* formula. I deviate from the text above and use as numbering of the weekdays Monday = 0, ..., Sunday = 6. This way the *Julian day number* is ≡ to the weekday. The subroutine `julian` calculates the Julian day number.

For positive y we have in REXX `c = y % 100` (integer divide), and the remainder is `g = y // 100`. This is false for negative y, when REXX yields `y//100 = -[-y/100]` instead. C suffers from the same wrong floor function as the other programming languages: the Euclidean algorithm gives the formula $a = q \cdot b + r$ with $q = [a/b]$ and $0 \le r < b$ for positive b. If a is negative the C function a/b gives the wrong result except if b divides a. The routines `quot` and `mod` correct the handling of the floor function for negative arguments (and I allow for negative years).

These implementations have been run on various platforms:

- REXX on VM, OS/2, DOS, Windows, Linux
- Perl on Linux, AiX, Windows
- C on Windows and Linux.

B.2. **Program listing** `gauss.c`**.**

```
/* ————————————————————————————————————————————————————————————*

   Module:        gauss.c
```

```
   Description:

        This program explores Gauss' formula for weekday calculation
        in the proleptic Gregorian Calendar.

        For a detailed explanation see my calendar article:
        https://berndt-schwerdtfeger.de/wp-content/uploads/pdf/cal.pdf

        Input:  year-mm-dd [offset] (ISO 8601 Format)

        Output: Weekday, date, Julian Calendar date, day of the year,
                week of the year, Julian day number, Unix day number

   Subroutines: date, offset, weekday, julian, leap, quot, mod

   Sample: gauss 2010-01-0 +120
        Fri 2010-04-30, JC 2010-04-17, D# 120 W# 17 J# 2455317 X# 14729

   ------------------------------------------------------------------------
```



```
   ------------------------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>

// set constants

char *wd[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};
int em[13] = {0,0,31,59,90,120,151,181,212,243,273,304,334};
const int JULIAN = 0;
const int GREGOR = 1;

// prototype statements for functions

void date(int,long,long*,int*,int*);  // type, offset, year, month, day
long offset(long,int,int);            // = offset into year y
int  weekday(long,int,int);           // = {0|1|2|3|4|5|6}, 0 = Monday
long julian(long,int,int);            // = number of days since JC -4712/01/01
```

```
int  leap(int,long);                    // = 1 if year is a leap year
long quot(long,long);                   // = [a/b] (corrected: works if a < 0)
long mod(long,long);                    // = a - quot(a,b)*b

// ------------------------------------------------------------------------
// main program
// ------------------------------------------------------------------------

int main(int argc, char *argv[]){

   if (argc == 1 || *argv[1] == '?'){
      printf("\ngauss v1.4, 2010-12-03\n");
      printf("-- Copyright (C) 2009-2010 Berndt E. Schwerdtfeger --\n\n");
      printf("  Input:      year-mm-dd [offset] \n\n");
      printf("  Output:     Weekday, date, Julian Calendar date, day of the year,\n");
      printf("              week of the year, Julian day number, Unix day number\n\n");
      printf("  Sample: gauss 2010-01-00 +120\n");
      printf("  Fri 2010-04-30, JC 2010-04-17, D# 120 W# 17 J# 2455317 X# 14729\n\n");

     return EXIT_SUCCESS;               // end the program
   }                                    // argc > 1 here

   int  w,m,d,jm,jd;
   long y,jy,j,x,n=0;
   char line[80];

   sscanf(argv[1],"%ld-%d-%d",&y,&m,&d);

   if (argc == 3)
      sscanf(argv[2],"%ld",&n);         // get the offset
   else if (argc > 3) {
      printf("Too many parameters !\n");
      return EXIT_FAILURE;              // end the program
   }

   while(1)
   {
      n += offset(y,m,d);               // correct the offset
      date(GREGOR, n, &y, &m, &d);      // correct y, m, d
      j = julian(y,m,d);                // set Julian day number
      x = j - 2440588;                  // set Unix day number
      jy = -4712;                       // set initial julian year
      date(JULIAN, j+1, &jy, &jm, &jd); // correct jy, jm, jd
      n = offset(y,m,d);                // set offset in this year
      w = weekday(y,m,d);               // set weekday (Gauss)

      printf("%s %ld-%02d-%02d, JC %ld-%02d-%02d, ",wd[w],y,m,d,jy,jm,jd);
      printf("D# %03d W# %02d J# %ld X# %ld\n",n,(n-w+9)/7,j,x);
      fgets(line,sizeof line,stdin);    // read next
      if (line[0]=='\n')
         break;
      n = 0;
```

```
      sscanf(line,"%ld-%d-%d_%ld",&y,&m,&d,&n);
   }                                    // end of while loop
   return EXIT_SUCCESS;                 // exit the program
};

// ------------------------------------------------------------------------
// subroutines: date, offset, weekday, julian, leap, quot, mod
// ------------------------------------------------------------------------
void date(int c, long n, long* y, int* m, int* d){

   int i;
   while (n > 365 + leap(c,*y)){        // if offset larger than # of
      n -= 365 + leap(c,*y);            // ... days in a year
      *y += 1;                          // ... find the correct year
   }

   while (n <= 0){                      // if offset is negative
      *y -= 1;                          // ... find the correct year
      n += 365 + leap(c,*y);            // ... and offset
   }

   i = leap(c,*y);                      // adjust for leap day
   *m = 12;
   while (n <= em[*m] + i){             // searching for the month
      *m -= 1;
      if (*m<3) i=0;
   }
   *d = n - em[*m] - i;                 // getting the day
}

// ------------------------------------------------------------------------
long offset(long y, int m, int d){

   int x = d + em[m];                   // offset into this year
   if (m > 2)
      x += leap(GREGOR,y);              // adjust for leap day
   return x;
}

// ------------------------------------------------------------------------
int weekday(long y, int m, int d){

   if (m < 3)
      y -= 1;

   long c = quot(y,100);
   int g = mod(y,100);
   int f = 5 * mod(c,4);
   int e = em[m];
   if (m > 2)
      e -= 1;
   return (-1 + d + e + f + g + g/4)%7; // Gauss' formula
```

```
}

// ------------------------------------------------------------------------
long julian(long y, int m, int d){

   long x = 0;
   if (m < 3) {
      y -= 1;
      x = 365;
   }
   long c = quot(y,100);
   int g = mod(y,100);
   x += em[m];
   return 1721060 + d + x + 365*y + quot(c,4) + 24*c + g/4;
}

// ------------------------------------------------------------------------
int leap(int c, long y){

   int i = 0;
   if (y%4 == 0)
      i = 1;
   if (c == GREGOR)
      if (y%100 == 0)                    // if century ..
         i -= (y%400 != 0);              // .. adjust
   return i;
}

// ------------------------------------------------------------------------
long quot(long a, long b){

   long x;
   x = a/b;
   if (a < 0 )                           // for negative numerator ..
      x -= (a%b != 0);                   // .. if remainder, subtract 1
   return x;
}

// ------------------------------------------------------------------------
long mod(long a,long b){
   return a - quot(a,b)*b;               // remainder always positive
}
```

B.3. **Program listing** `gauss.rexx`**.**

```
/* set constants and initialize variables */

   d. = 31                          /* default value for days in a month    */
   d.2 = 28; d.4 = 30; d.6 = 30; d.9 = 30; d.11 = 30

   e. = 0

   Do m = 1 to 12
```

```
      e.m = e.0                  /* number of days in previous months    */
      e.0 = e.m + d.m            /* simply sum them up                   */
   end

   d.0  = "Mon"; d.1  = "Tue"; d.2  = "Wed"; d.3  = "Thu";
   d.4  = "Fri"; d.5  = "Sat"; d.6  = "Sun";

/* get the arguments */

   Arg date n .
   If date = '' then signal help

   do forever

      If substr(date,1,1) = '-'
         then do
              parse var date '-' y '-' m '-' d
              y = -y
           end
         else parse var date y '-' m '-' d
      If n = '' then n=0
      If m = '' | m < 1 | m > 12 | d = '' then signal error

      n = n + offset(y,m,d)          /* get correct offset              */
      date = date(n,y)               /* get correct date                */
      If substr(date,1,1) = '-'
         then do
              parse var date '-' y '-' m '-' d
              y = -y
           end
         else parse var date y '-' m '-' d
      j = julian(y,m,d)              /* get Julian day number           */
      x = j - 2440588                /* get Unix day number             */
      jdate = jdate(j+1,-4712)       /* get Julian date                 */
      n = right(offset(y,m,d),3,0)   /* get offset in this year         */
      w = weekday(y,m,d)             /* get weekday (Gauss)             */

      say d.w date || jdate 'D#' n 'W#' right((n-w+11)%7,2,0) 'J#' j 'X#' x

      Pull date n .
      If date = '' then exit
   end

return

error:
   do
      say 'ERROR: month and day must not be empty'
      say '       and month in the range 1 .. 12'
   end
help:
   do
```

```
      say ' '
      say 'Gregorian Calendar (since 1582-10-15)'
      say ' '
      say ' Input:      year-month-day [offset]'
      say ' '
      say ' Output:     Weekday, date, Julian Calendar date, day of the year,'
      say '             week of the year, Julian day number, Unix day number'
      say ' '
      say ' Example: gauss.rexx 2018-11-28 -16'
      say ' Mon 2018-11-12, JC 2018-10-30, D# 316 W# 46 J# 2458435 X# 17847'
      say ' '
      exit
   end


/* ---------------------------------------------------------------------- *
 * Subroutines
 *   date(offset, year) = 'y-m-d'          Gregorian Calendar date
 *   jdate(offset, year) = 'JC y-m-d'      Julian Calendar date
 *   offset(y,m,d) = offset into year y
 *   weekday(y,m,d) = {0|1|2|3|4|5|6} (Gauss formula) 0 = Monday
 *   julian(y,m,d) = integer, number of days since JC -4712-01-01
 *   leap(year) = 1 if year is a leap year
 *   div(a,b) = int a/b  (corrected integer divide, works if a < 0)
 *   mod(a,b) = a - div(a,b)*b
 * ---------------------------------------------------------------------- */

date: procedure expose e.

   Arg d , y

   y = y + 0                            /* remove trailing blank, maybe */

   Do while d > 365 + leap(y)           /* if offset larger than # of   */
      d = d - 365 - leap(y)             /* ... days in a year           */
      y = y + 1                         /* ... find the correct year    */
   end

   Do while d <= 0                      /* if offset is negative        */
      y = y - 1                         /* ... find the right year      */
      d = d + 365 + leap(y)             /* ... and offset               */
   end

   l. = leap(y); l.1 = 0; l.2 = 0       /* adjust for leap day          */
   m = 12
   Do while d <= e.m + l.m              /* searching for the month      */
      m = m - 1
   end

   d = right(d - e.m - l.m , 2 , 0)
   m = right(m,2,0)
```

```
   return y || '-' || m || '-' || d


jdate: procedure expose e.

   Arg d , y

   Do while d > 365 + (y//4=0)          /* if offset larger than # of   */
      d = d - 365 - (y//4=0)            /* ... days in a year           */
      y = y + 1                         /* ... find the correct year    */
   end

   Do while d <= 0                      /* if offset is negative        */
      y = y - 1                         /* ... find the right year      */
      d = d + 365 + (y//4=0)            /* ... and offset               */
   end

   l. = (y//4=0); l.1 = 0; l.2 = 0      /* adjust for leap day          */
   m = 12
   Do while d <= e.m + l.m              /* searching for the month      */
      m = m - 1
   end

   d = right(d - e.m - l.m , 2 , 0)
   m = right(m,2,0)

   return ',␣JC' y || '-' || m || '-' || d || ','


offset: procedure expose e.

   Arg y , m , d

   l. = leap(y); l.1 = 0; l.2 = 0       /* adjust for leap day          */
   m = m + 0

   return d + e.m + l.m                 /* offset into this year        */


weekday: procedure expose e.

   Arg y , m , d

   If m < 3 then y = y - 1

   c = div(y,100)
   g = mod(y,100)
   f = 5 * mod(c,4)
   m = m + 0
   e = e.m
   If m > 2 then e = e - 1
```

```
    return ( -1 + d + e + f + g + g % 4 ) // 7          /* Gauss' formula */


julian: procedure expose e.

    Arg y, m, d

    e = 0
    If m < 3 then
    do
      y = y - 1
      e = 365
    end
    c = div(y,100)
    g = mod(y,100)
    m = m + 0
    e = e + e.m

    return 1721060 + d + e + 365*y + div(c,4) + 24*c + g%4


leap: procedure

    Arg y

    c = div(y,100)
    g = mod(y,100)

    return ( g \= 0 & g // 4 = 0 ) | ( g = 0 & c // 4 = 0 )


div: procedure

    Arg a , b

    x = a % b
    If a < 0 then x = x - ( a // b \= 0 )

    return x


mod: procedure

    Arg a , b

    return a - div(a,b)*b
```

INDEX