UNIVERSIDADE FEDERAL DO CARIRI
CENTRO DE CIÊNCIAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA EM REDE NACIONAL

CÁSSIO GOMES DE LIMA

DETERMINANDO A DATA DAS FESTIVIDADES MÓVEIS ATRAVÉS DA LINGUAGEM C++

JUAZEIRO DO NORTE
2017

CÁSSIO GOMES DE LIMA

DETERMINANDO A DATA DAS FESTIVIDADES MÓVEIS ATRAVÉS DA LINGUAGEM C++

Dissertação (Mestrado) apresentada ao Programa de Pós-graduação em Matemática em Rede Nacional (PROFMAT), do Departamento de Matemática da Universidade Federal do Cariri, como parte dos requisitos necessários para a obtenção do título de Mestre em Matemática. Área de concentração: Ensino de Matemática.

Orientador: Prof. Dr. Paulo César Cavalcante de Oliveira

JUAZEIRO DO NORTE
2017

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Cariri
Sistema de Bibliotecas

L732d Lima, Cássio Gomes de.
Determinando a data das festividades móveis através da linguagem C++/ Cássio Gomes de Lima. – 2017.
54 f.: il.; color.; enc. ; 30 cm.

Dissertação (Mestrado) – Universidade Federal do Cariri, Centro de Ciências e Tecnologia – Programa de Pós-graduação em Matemática em Rede Nacional, Juazeiro do Norte, 2017.
Área de Concentração: Ensino de Matemática.

Orientação: Prof. Dr. Paulo César Cavalcante de Oliveira.

1. Aritmética modular. 2. Fórmula de Gauss. 3. Linguagem C++. I. Oliveira, Paulo César Cavalcante de. II. Título.

CDD 513.1

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO CARIRI
CENTRO DE CIÊNCIAS E TECNOLOGIA
MESTRADO PROFISSIONAL EM MATEMÁTICA EM REDE NACIONAL - PROFMAT

Determinando a Data das Festividades Móveis Através da Linguagem C++

Cássio Gomes de Lima

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Matemática em Rede Nacional – PROFMAT do Centro de Ciências e Tecnologia da Universidade Federal do Cariri, como requisito parcial para obtenção do Título de Mestre em Matemática. Área de concentração: Ensino de Matemática

Aprovada em 26 de julho de 2017.

Banca Examinadora

Paulo César Cavalcante de Oliveira

Prof. Dr. Paulo Cesar Cavalcante de Oliveira - URCA

Orientador

Job Saraiva Furtado Neto

Prof. Dr. Job Saraiva Furtado Neto - UFCA

Flávio França Cruz

Prof. Dr. Flávio França Cruz – URCA

À Francisca, Felipe e Cícero (in memoriam).

AGRADECIMENTOS

A Deus, por dar-me saúde, vontade e mais essa oportunidade, no sentido de tornar-me sempre melhor, principalmente como pessoa.

À Francisca Gomes da Silva Lima, minha amorosa mãe, sem a qual, dificilmente eu teria chegado até aqui, e tamanho esforço fez para, praticamente sozinha, proporcionar a mim e a meu irmão o pão, a educação e o modelo de fé que temos.

A Felipe Gomes de Lima, meu eterno e valoroso irmão, amigo, leal e companheiro, apoio desde sempre e retidão como nunca.

À Marília Muryel Estevam Alves, minha amada, que tanto me ajudou e me deu garra, e pelo carinho e dedicação nos momentos alegres e tristes.

Ao meu orientador Paulo César Cavalcante de Oliveira, pelo apoio, pelas indicações e que, desde o tempo da graduação, mesmo não tendo sido meu professor, é exemplo de competência e eficiência.

Aos professores Flávio França Cruz e Job Saraiva Furtado Neto, pelo valoroso tempo investido e pelas relevantes sugestões no trabalho.

À professora Maria Silvana Alcântara Costa, coordenadora do programa nesta instituição, pelo apoio desde antes do curso até, tenho certeza, quando sentir que não há mais necessidade, e pelo compromisso e responsabilidade com esta turma e com a instituição.

Aos professores Plácido Francisco de Assis Andrade, Clarisse Dias de Albuquerque, Juscelino Pereira Silva e Francisco Calvi da Cruz Júnior, pelas boas orientações em sala e fora dela.

Aos professores Mário de Assis, Zelalber Gondim, Carlos Alberto (Carlão), Luiz Antônio (Luizin), Luciana Maria, Bárbara Paula, Alexsandro Coelho, Pedro Ferreira, José Tiago, Kátia Pires, Valéria Gerônimo, Ricardo Rodrigues (além do já citado Flávio França) e outros (não citados e não menos importantes), da graduação, e também a Guttenberg Sergistótanes, Marcos Aurélio (o de Matemática e o de Inglês), Gilberto Alves, Edilberto Gonçalves, Neuma Xenofonte, Ernesto Luiz (in memoriam), Jorgeana Jorge, Luis Edvan, Marcos Chaves, Ana Paula (a de Química e a de Português) e Cláudio Dantas, pela excelente formação acadêmica, exemplo e inspiração, formadores da sólida base sobre a qual me construi neste meio.

Aos meus companheiros de curso, pela presença e apoio, por todos os momentos de riso e cooperação, pelas novas amizades, e especialmente a Helano, Rommel e Irlã, pela força, palpites, alegrias e amizade. Também são meus professores.

A todos, meus eternos professores.

Ao PROFMAT, pelo aperfeiçoamento pessoal, intelectual e profissional, eterno, e à CAPES, pelo apoio financeiro com a bolsa de auxílio.

"A nobreza e a excelência de uma ciência devem ser julgadas pela certeza das demonstrações que ela usa. Sem dúvida as disciplinas matemáticas estão em primeiro lugar entre todas as outras."

Cristóvão Clávio (1538-1612)

RESUMO

O calendário como o conhecemos hoje (*gregoriano*) sofreu algumas alterações importantes, e muita resistência de países, reinos e até mesmo de bispos de regiões mais afastadas, até ser aceito como uno, reconhecendo os erros do tomado anteriormente (*juliano*). Até a reforma promulgada pelo Papa Gregório XIII, apenas seguia-se a regra de acrescentar um dia a fevereiro a cada 4 anos, e isso gerou uma defasagem, por exemplo, na data de início das estações, que não mais coincidiam com solstícios e equinócios astronômicos, fazendo com que a Páscoa fosse festejada fora do período determinado no Concílio de Nicéia, e conseguinte, outras festividades móveis. Após 1582, o proclamado Ano da Reforma, houve desavença e desestruturação na Igreja Católica, e também desta com nações pelo mundo. Tanto é que Grécia e Turquia, por exemplo, mantiveram-se com o antigo calendário até a última década de 20. A influência da igreja garantiria, a nível mundial, a adesão ao novo calendário, e, além disso, a própria evolução da sociedade, cada vez mais interligada e interdependente, levaria à adoção do mesmo, cedo ou tarde. O cálculo dos anos bissextos é bem claro e simples, até na Bula Papal do próprio Gregório, e assim como ele, a regra da determinação do domingo de Páscoa são facilmente encontrados, e este último, com o surgimento e desenvolvimento da informática, acelerou ainda mais tal cômputo. Este trabalho visa, sob o olhar da problemática das datas móveis e a história do calendário gregoriano, apresentar a Fórmula de Gauss que determina a data da Páscoa de qualquer ano após 1600, e "traduzi-la" numa linguagem de programação C++. Para isso, conterá os tópicos necessários da aritmética para a construção do mesmo, em especial a aritmética modular, além do instrumental básico de programação para a confeccção deste programa. Logo, um aluno de ensino médio com conhecimento básico ou em um curso técnico/profissionalizante de informática, com a leitura deste texto, terá os fundamentos para a implementação da regra em outras linguagens (até em planilhas eletrônicas), baseando-se na matemática por trás desse processo.

Palavras-chave: Aritmética Modular. Fórmula de Gauss. Linguagem C++.

RESUMÉ

Le calendrier comme nous le connaissons aujourd'hui (*grégorien*) a subi des changements importants et beaucoup de résistance de la part des pays, royaumes et même des évêques des régions plus éloignées, pour être accepté comme l'une, en reconnaissant les erreurs du prise précédemment (*julien*). Jusqu'à la réforme promulguée par Pape Grégoire XIII, seulement il a été suivi la règle d'ajouter un jour à Février chaque 4 ans, et cela a créé un lag, par exemple, au début des saisons, qui ne coïncident plus avec solstices et équinoxe astronomique, ce qui provoque la fête de Pâques célébrée en dehors de la période spécifiée au Conseil de Nice, et donc d'autres festivals mobiles. Après 1582, a proclamé l'année de la Réforme, il y avait désaccord et la rupture dans l'Eglise catholique, et cela aussi avec certains pays du monde. Si bien que Grèce et Turquie, par exemple, est resté avec l'ancien calendrier jusqu'à la dernière décennie à 20. L'influence de l'église garantirait, partout dans le monde, le respect du nouveau calendrier, et en plus, l'évolution de la société, de plus en plus interconnecté et interdépendant, conduirait à l'adoption du même, tôt ou tard. Le calcul des années bissextiles est clair et simple, même dans la bulle papale de lui-même Grégoire, et comme il, la règle de détermination du dimanche de Pâques sont faciles à trouver, et ce dernier, avec l'émergence et le développement des technologies de l'information, aussi accélérée tel calcul plus. Le présent document vise, sous le regard du problème des dates mobiles, et l'histoire du calendrier grégorien, présente la formule Gauss qui détermine la date de Pâques dans une année après 1600, et la-traduisez pour une langage de programmation C ++. Pour cela, contiendra les sujets arithmétiques nécessaires à la construction du même, en particulier l'arithmétique modulaire, en plus de l'instrumental de programmation de base pour la fabrication de ce programme. Ainsi, un élève du secondaire ayant une connaissance de base ou un cours de technicien en informatique, à la lecture de ce texte, aura le fondement de la mise en œuvre de la règle dans d'autres langues (même dans des feuilles de calcul), sur la base des mathématiques derrière ce processus.

Mots-clés: Arithmétique Modulaire. Formule Gauss. Langage C++.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Pessach(`passagem`, em hebraico): festa da tradição judaica, em comemoração à libertação do povo hebreu da escravidão no Egito, que teria acontecido em 14 *Nisan* de 1440 a.C. O calendário judaico começa na primavera, onde Nisan é o primeiro mês, que, como os outros, sempre começa na lua nova, portanto o décimo quarto dia sempre é lua cheia. Conta-se no cristianismo que a morte de Jesus aconteceu na semana do Pessach. Daí a associação deste com a *Páscoa*, desde o início do cristianismo, e de onde deriva o próprio nome [6].

A questão é que a Páscoa (celebração da Ressurreição de Jesus) era comemorada nas igrejas da Ásia Menor, como na grande Éfeso (hoje *Igreja Católica Ortodoxa*), juntamente com os judeus, exatamente no 14 Nisan, independente do dia da semana, ao passo que as igrejas de Roma e Alexandria (poderosas na época[1]), entre outras igrejas ocidentais e orientais, o faziam no primeiro domingo após esta data, pela tradicional importância deste dia após *Jesus Cristo*.

Quanto ao calendário, o imperador romano Júlio César implantou, a partir de 46 a.C., o ano como tendo 12 meses, com o mês *februarius* tendo 29 dias, e um 30º dia a cada 4 anos. No império de Otávio (posteriormente chamado de *Augusto César*) foi dado ao mês *quintilis* o nome de *julius*, em homenagem ao antigo imperador. Em seguida, convencendo o senado, conseguiu que o mês *sextilis* fosse trocado por *augustus*, em sua homenagem, fazendo com que se acrescentasse a este mais um dia (para, assim como o mês de Júlio César, tivesse 31), que seria retirado de *februarius*. Isso é mostrado na **Tabela 1**.

No ano 325 d.C., o imperador romano Constantino convocou o Concílio de Nicéia [2], presidido pelo bispo Ósio de Córdoba (Hispânia) em nome do velho Papa Silvestre I, onde, dentre outros temas de interesse da fé cristã, foi definido que todas as igrejas deveriam celebrar a Páscoa no mesmo dia, num domingo, mas não mais consultando o calendário judaico. Ficou definido então que deveria ser celebrada "*no domingo a seguir à lua cheia, depois de 21 de Março (Equinócio*[3] *de Primavera), o que dá um domingo entre* 22 *de Março e* 25 *de Abril*" [13].

Todos as igrejas estavam cumprindo o determinado em Nicéia, porém com o passar dos séculos foi-se vendo uma defasagem de 10 dias entre a data de início da

[1] À época do Concílio de Nicéia, a capital do império romano era Constantinopla, fundada pelo então imperador Constantino, e recentemente movida da antiga Roma. Isso justifica o prestígio desta perante as igrejas durante o concílio.

[2] Hoje Iznik, cidade turca no arredores de Istambul.

[3] O equinócio é o fenômeno onde dia e noite tem a mesma duração, em todos os pontos da Terra. No hemisfério norte, quando é equinócio de Primavera, no hemisfério sul é equinócio de Outono, e vice-versa.

Calendário de Júlio César		Depois de Augusto César	
Mês	Dias	Mês	Dias
Januarius	31	Januarius	31
Februarius	29/30	Februarius	28/29
Martius	31	Martius	31
Aprilis	30	Aprilis	30
Maius	31	Maius	31
Junius	30	Junius	30
Quintilis	31	Julius	31
Sextilis	30	Augustus	31
September	31	September	30
October	30	October	31
November	31	November	30
December	30	December	31

Tabela 1 – Mudanças no calendário, segundo Júlio César e Augusto César.
Fonte: [9].

primavera e o equinócio propriamente dito, devido ao calendário de Júlio César, gerando assim uma grande confusão. A solução já vinha sendo debatida a alguns séculos por intelectuais e nobrezas, mas nunca definida por líder algum. O problema de retirar dias para voltar o equinócio para a data certa, exigia tamanho poder e influência, que seria "necessária a autoridade de um papa para implantar a mudança (assim como já havia sido o papel de um ditador romano)" [4].

E assim o fez o Papa Gregório XIII, que para tentar consertar tudo, convidou estudiosos para propor e analisar alternativas. O médico e professor italiano Luigi Lilio, com a ajuda de seu irmão Antonio, sugeriu que se retirassem 10 dias para voltar o equinócio à data normal, além de alterar como seriam determinados os anos bissextos, para não mais se ter esse problema no futuro. Segundo ele, $365,2425$ dias era uma aproximação bem melhor para o ano trópico[4], diferente dos $365,25$ dias do calendário juliano (com mais de 11 minutos de diferença). Isso resulta em 365 dias, 5 horas, 49 minutos e 12 segundos; são apenas $26,8$ segundos além do ano trópico!

Para chegar a esse valor, deveria-se continuar a contagem dos anos bissextos, excluindo os anos seculares (múltiplos de 100), com exceção do múltiplos de 400, que continuariam com um dia a mais:

$$365 + \frac{1}{4} - \frac{1}{100} + \frac{1}{400} = 365,2425. \tag{1.1}$$

E assim, com o "aval" do matemático e astrônomo jesuíta Cristóvão Clávio, Gregório publicou a bula papal *Inter Gravissimas* em 24 de fevereiro de 1582, determinando: que a próxima quinta-feira 4 de outubro seria sucedida pela sexta-feira 15 de outubro; que o ano 1600 continuaria bissexto; que os 3 anos seculares seguintes não o

[4] Tempo que a Terra leva para completar sua órbita.

seriam (1700, 1800 e 1900), mas o ano 2000 voltaria a sê-lo, tomando esse mesmo padrão adiante; além de outras diretrizes sociais e eclesiásticas decorrentes dessas mudanças [11]. Enviou para todas as igrejas, ao passo que aquelas da Ásia Menor, não reconhecendo a primazia do papa, continuaram seguindo o calendário juliano, e o fazem até hoje. Já os países foram cedendo às recomendações. Alguns católicos, imediatamente, como Espanha, França e Portugal (e Brasil, por consequência), e outros com o passar dos anos, sendo registrado que Grécia e Turquia (predominantemente católicos ortodoxos) adotaram-nas em 1923 e 1926, respectivamente, e por último a República Popular da China, com sistema próprio (o antiquíssimo *calendário chinês*).

O objetivo central deste trabalho é apresentar um algoritmo que calcula a data móvel da Páscoa, que depende do *plenilúnio*[5], do domingo, entre todos os dias da semana, e também do equinócio da primavera (no hemisfério norte!). Para isso, na introdução mostrou-se o motivo histórico do cômputo e suas particularidades, além da sua contextualização com a reforma do calendário promulgada pelo Papa Gregório XIII, que influencia no cálculo desta data, pela presença (ou não) do 366º dia.

No capítulo 2 será apresentado todo o embasamento necessário para a construção do algoritmo que depende apenas de Teoria dos Números, mais especificamente de um tópico: *congruências*, ou, como às vezes é chamada, *aritmética dos restos*. Isso se explica pelo caráter cíclico dos cálculos envolvidos na determinação da tal data.

Já o capítulo 3 abordará os pormenores da linguagem de programação C++, desde o básico para um rotina simples até uns poucos comandos mais sofisticados que serão usados na implementação do algoritmo do cálculo da Páscoa. Lembrando-se que por se tratar de uma lógica de programação, acrescentando poucas linhas é possível incrementar um algoritmo simples, e torná-lo mais completo e eficiente.

O passo a passo da construção de um programa nessa linguagem que retorne a data da Páscoa de um ano qualquer (a partir de 1600) será ilustrado no capítulo 4. Como esta data é referência para a determinação de outras festividades religiosas[6] (fixas, com relação a ela), facilmente é possível determiná-los por consequência da praticidade da linguagem usada e do simples cálculo que é feito para tal. Os comandos básicos de C++ descritos aqui, os exemplos apresentados e discutidos e o fato do objeto de pesquisa (o calendário) ser tão concreto, por estar tão atrelado à vida cotidiana, gera interesse por parte dos alunos do ensino médio, podendo ser usado sem problemas por aqueles estudantes de escolas técnicas ou profissionalizantes do curso de informática.

[5] Lua cheia.

[6] Daí sua importância para a Igreja.

2 Noções Básicas de Teoria dos Números

Uma parte importante da teoria dos números é a que estuda fenômenos cíclicos, pois a natureza, física e humana, frequentemente se manifesta em situações desse tipo. Por exemplo: *Numa rodoviária, Pedro que toma um ônibus às 10 horas da manhã é informado que o tempo estimado de viagem é de 16 horas, ao que imediatamente ele calcula a sua chegada às 2 da manhã do dia seguinte.* Ora, como somar $10 + 16$ e obter como resultado 2? Simples: usando a aritmética dos restos, ou como também é conhecida, *aritmética modular*. Para iniciar a abordagem do tema, tratemos da divisibilidade entre inteiros.

2.1 DIVISIBILIDADE EM $\mathbb{Z}$

É sabido que nem todo número natural (ou inteiro) é divisível por outro. Neste caso, dizemos que essa divisão deixa resto (*diferente de zero*). Quando a divisão é possível, usamos a seguinte definição:

Definição 1 *Dados dois números inteiros a e b com $a \neq 0$, diz-se que a* divide *b (ou $a|b$) quando existir um $q \in \mathbb{Z}$ tal que $b = a \cdot q$. Sendo assim, a é* divisor *de b, ou b é* múltiplo *de a, ou ainda b é* divisível *por a.*

Quando a não divide b, ou, em outras palavras, quando não existir $q \in \mathbb{Z}$ tal que $b = aq$, representa-se por $a \not| b$. Ainda, se q existe, ele é chamado de *quociente* de b por a.

Exemplo 1 1, 2, 3, 4, 6 e 12 são todos divisores de 12, pois

$$\begin{aligned} 12 &= 1 \cdot 12, \\ 12 &= 2 \cdot 6, \\ 12 &= 3 \cdot 4. \end{aligned}$$

Logo, temos que $1|12$, $2|12$, $3|12$, $4|12$, $6|12$ e $12|12$. Entretanto, $5 \not| 12$, pois não existe $q \in \mathbb{Z}$ tal que $12 = 5q$.

$\diamond$

Note-se que

$$a \in \mathbb{Z}^* \, , \, a|b \Rightarrow -a|b$$

pois

$$b = aq = (-a)(-q).$$

Isso significa que para cada divisor de um número, o simétrico[1] daquele também o é. Com isso, e partindo ainda da **Definição 1**,temos:

[1] Mesmo valor absoluto, mas com sinal oposto.

Proposição 2.1.1 *A relação de divisibilidade em $\mathbb{Z}$ é:*

i) reflexiva; e

ii) transitiva.

Demonstração:
i) $a = a \cdot 1 \Rightarrow$ a divide ele mesmo.
ii) Significa que se $a|b$ e $b|c \Rightarrow a|c$. De fato:

$$a|b \Rightarrow b = aq_1$$

$$b|c \Rightarrow c = bq_2 = aq_1q_2 \Rightarrow a|c.$$

■

Entretanto, essa relação não é simétrica, como por exemplo $2 \,|\, 10$ mas $10 \not| \, 2$. Além destas propriedades, temos:

Proposição 2.1.2 *Sejam $a, b, c \in \mathbb{Z}$. Assim:*

i) $1|a$ *e* $a|0$.

ii) se $a|1$, *então* $a = \pm 1$.

iii) se $a|b$ *e* $c|d$, *então* $ac|bd$.

iv) se $a|b$ *e* $b|a$, *então* $a = \pm b$.

v) se $a|b$, *com* $b \neq 0$, *então* $|a| \leq |b|$.

Demonstração:
i) $a = 1 \cdot a$ e $0 = a \cdot 0$.
ii) $a|1 \Rightarrow 1 = aq \Rightarrow a = q = 1$ ou $a = q = -1$.
iii) De fato, $a|b$ e $c|d \Rightarrow b = aq_1$ e $d = cq_2$. Então $bd = aq_1cq_2 = (ac)(q_1q_2) \Rightarrow ac|bd$.
iv) De fato, $a|b$ e $b|a \Rightarrow b = aq_1$ e $a = bq_2 = aq_1q_2$. Então $q_1 = q_2 = 1$ ou $q_1 = q_2 = -1 \Rightarrow a = \pm b$.
v) De fato, como $a \,|\, b$, com $b \neq 0 \Rightarrow b = aq$, o que implica que $|b| = |a||q|$. Só que $q \neq 0 \Rightarrow |q| \geq 1 \Rightarrow |b| \geq |a|$.

■

Por fim, uma das mais importantes propriedades da divisibilidade:

Proposição 2.1.3 *Sejam a, b e c inteiros, tais que $a \neq 0$. Se a divide b e c, então divide qualquer combinação linear*[2] *entre eles, isto é:*

[2] Em referência ao caráter linear (linha reta) da expressão formada quando comparada a uma constante, e representada no sistema de coordenadas xOy.

$$a|b \text{ e } a|c \Rightarrow a|(bx+cy),\ \forall x, y \in \mathbb{Z}.$$

Demonstração:
De $a\,|\,b$ e $a\,|\,c$, temos que $b = aq_1$ e $c = aq_2$, com $q_1, q_2 \in \mathbb{Z}$. Com isso, para quaisquer $x, y \in \mathbb{Z}$, tem-se que

$$bx + cy = aq_1x + aq_2y = a(q_1x + q_2y) \Rightarrow a|(bx+cy).$$

■

Note que, por esta propriedade, se $a\,|\,b_i$, com $i = 1, 2, \ldots, n$, para quaisquer $x_i \in \mathbb{Z}$, temos que

$$a|(b_1x_1 + b_2x_2 + \ldots + b_nx_n).$$

Exemplo 2 ([1]) Temos que, se $a\,|\,(2x-3y)$ e $a\,|\,(4x-5y)$, então $a\,|\,y$. De fato, pela **Proposição 2.1.3**, temos que

$$a|[i(2x-3y) + j(4x-5y)] \quad ,\forall i, j \in \mathbb{Z}.$$

Se a divide qualquer combinação linear, ou seja, para quaisquer i e j, divide, em especial, para $i = -2$ e $j = 1$:

$$a|[-2(2x-3y) + (4x-5y)] \Rightarrow a|y.$$

◇

Para complementar esta seção, mais outra importantíssima propriedade:

Proposição 2.1.4 *Sejam $a, b \in \mathbb{Z}$ e $n \in \mathbb{N}$. Nessas condições, $(a-b)|(a^n - b^n)$.*

Demonstração:
Vamos usar indução sobre n. A princípio, pela **Proposição 2.1.2(i)**, $n = 0$ é válido pois

$$(a-b)|(a^0 - b^0) \Rightarrow (a-b)|0.$$

Suponhamos agora, por hipótese de indução, que $(a-b)|(a^n - b^n)$. Temos que

$$a^{n+1} - b^{n+1} = aa^n - bb^n = aa^n - ba^n + ba^n - bb^n = (a-b)a^n + b(a^n - b^n).$$

Sabemos que $(a-b)|(a-b)$ e $(a-b)|(a^n-b^n)$, por hipótese. Logo, da **Proposição 2.1.3**, temos que $(a-b)|(a^{n+1} - b^{n+1})$, e pela indução, para todo $n \in \mathbb{N}$.

■

2.2 DIVISÃO EUCLIDIANA E MÁXIMO DIVISOR COMUM

Para tratarmos desse importante tópico, será introduzido no texto o *Princípio da Boa Ordem*, mas sua demonstração, apesar de simples, será omitida, por não ser objetivo do texto. Para apreciação da mesma, consultar [1].

Teorema 2.2.1 (Princípio da Boa Ordem) *Todo subconjunto não-vazio de* $\mathbb{Z}^+$ *possui um menor elemento.*

A divisão euclidiana (que tem esse nome devido o vasto uso na obra-prima *Elementos*, de *Euclides*) é a base deste trabalho.

Teorema 2.2.2 (Divisão Euclidiana) *Sejam* a *e* b *dois inteiros, com* $a > 0$. **Existem únicos** *inteiros* q *e* r *tais que*

$$b = aq + r \quad , 0 \leq r < a.$$

Demonstração:
Suponhamos o conjunto $\mathcal{S}$ de inteiros não-negativos da forma $b - na$, com $n \in \mathbb{Z}$. $\mathcal{S}$ é não vazio, pois, como $a > 0 \Rightarrow a \geq 1$, e assim, para um $n = -|b|$, temos

$$b - na = b + |b|a \geq b + |b|$$

que é não negativo, e portanto, "possível" elemento de $\mathcal{S}$. Além do mais, pelo *Princípio da Boa Ordem*, existe um menor $b - na$. Seja ele

$$r = b - qa \quad ou \quad b = aq + r \quad , q \in \mathbb{Z}.$$

No mais, se $r \geq a$, então teríamos um $x \in \mathbb{N}$ tal que $r = a + x = b - qa$ o que implica

$$x = b - (q + 1)a < r$$

e portanto, contradição ao fato de r ser o *mínimo* de $\mathcal{S}$. Agora, mostremos que q e r são *únicos*. Suponhamos outros $q', r' \in \mathbb{Z}$ tais que

$$b = aq' + r' \quad , 0 \leq r' < a.$$

Assim

$$aq + r = aq' + r' \Rightarrow r - r' = (q' - q)a \Rightarrow a|(r - r').$$

Como $0 \leq r, r' < a$, isso implica que $|r - r'| < a$. Mas também $a\,|\,(r - r')$, então, pela **Proposição 2.1.2(v)**, deve ser

$$r - r' = 0 \Rightarrow r = r'$$

além de

$$q' - q = 0 \Rightarrow q' = q, \quad pois\ a \neq 0.$$

■

Dividindo então b por a, damos a q e r os nomes *quociente* e *resto*, respectivamente. É interessante ainda comentar que através desse algoritmo podemos definir um número inteiro como *par* ou *ímpar*, a seguir.

Definição 2 (Paridade) *Seja $b \in \mathbb{Z}$. Então*

i) se o resto da divisão de b por 2 for 0, então $b = 2q$ recebe o nome de "par";

ii) se o resto da divisão de b por 2 for 1, então $b = 2q + 1$ recebe o nome de "ímpar".

Generalizando essa definição, estabelecemos o seguinte corolário:

Corolário 2.2.1 *Seja $n \in \mathbb{N}$. Dado $a \geq 2$, sempre é possível expressá-lo, de forma única, como $n = ak + r$, onde $k, r \in \mathbb{N}$ e r é um dos números $0, 1, 2, \ldots, a-1$.*

Vamos aplicar isso num exemplo.

Exemplo 3 ([7]) De n inteiros consecutivos, um, e apenas um deles, é divisível por n. Pelo **Corolário 2.2.1**, cada um dos n números (distintos) é escrito, de forma única, como $nk + r$, com $r < n$. Como são consecutivos, cada um terá um valor distinto para r, e portanto, apenas um deles terá $r = 0$, isto é, é divisível por n, deixando resto 0.

◇

Exemplo 4 Quantos termos da *PA* $3, 8, 13, \ldots$ estão entre 84 e 427? Perceba que cada elemento dessa PA é um elemento da forma $5k - 2$, com $k = 1, 2, 3, \ldots$. Então conta-se essa quantidade de termos analisando qual o maior valor possível de k tal que $5k-2 < 427$ e daí subtrai-se do valor encontrado em $5k - 2 < 84$:

$$5k - 2 < 427 \Rightarrow k < 85,8 \quad e \quad 5k - 2 < 84 \Rightarrow k < 17,2.$$

Isso significa que, como k é natural, temos 85 termos da PA menores que 427, sendo 17 menores que 84, o que nos dá $85 - 17 = 68$ destes termos no intervalo $(84, 427)$. Como opção, poderíamos simplesmente ter adicionado 2 unidades a cada termo da PA e aos limites do intervalo, que a quantidade requisitada no problema se manteria, pois haveria apenas um deslocamento das *posições*; os elementos da "nova" PA seriam $5, 10, 15, \ldots$, entre 86 e 429. Assim, os elementos "coincidiriam" com os múltiplos positivos de 5. A estratégia seria calcular o quociente dos extremos do intervalo e 5, pela divisão euclidiana:

$$429 = 5 \cdot 85 + 4 \quad e \quad 86 = 5 \cdot 17 + 1.$$

Os múltiplos de 5 são

$$\underbrace{5 \cdot 1,\ 5 \cdot 2,\ 5 \cdot 3, \ldots, 5 \cdot 17 = 85}_{\leq 86},\ \underbrace{5 \cdot 18 = 90, \ldots, 5 \cdot 85 = 425}_{86 < n < 429},\ \underbrace{5 \cdot 86 = 430, \ldots}_{\geq 429}$$

concluindo assim como na solução anterior.

◇

Generalizando esse importante resultado, dados $a, c \in \mathbb{N}$, com $a < c$, o quociente da divisão euclidiana de c por a nos dá a quantidade de múltiplos positivos de a menores que c. Representaremos esse quociente (inteiro – e positivo) por $\left[\frac{c}{a}\right]$, que é a parte inteira do racional $\frac{c}{a}$. Isso demonstra a proposição abaixo, base para o **Capítulo 4**:

Proposição 2.2.1 *Dados $a, b, c \in \mathbb{Z}$, com $0 < a < b < c$, o número de múltiplos de a entre b e c é dado por:*

i) $\left[\frac{c}{a}\right] - \left[\frac{b-1}{a}\right]$, *$b$ incluso; ou*

ii) $\left[\frac{c}{a}\right] - \left[\frac{b}{a}\right]$, *$b$ excluso.*

A partir deste ponto precisaremos de uma das mais importantes definições da Matemática: *máximo divisor comum*. Sejam então a e b dois inteiros, não simultaneamente nulos. Dizemos que $d \in \mathbb{Z}^*$ é um divisor comum de a e b se $d|a$ e $d|b$. Baseado nisso:

Definição 3 (Máximo divisor comum) *Um número $d \in \mathbb{Z}_+^*$ é o* **máximo divisor comum** *de a e b se satisfaz as propriedades:*

i) d é um divisor comum de a e b; e

ii) sendo c um divisor comum de a e b, então $c|d$.

Assim, indica-se d por $\text{mdc}(a, b)$ ou apenas (a, b), notação usada neste trabalho. Decorre diretamente daí, por *(i)*, que d é um divisor comum a ambos os números e, por *(ii)*, que é o *maior* dentre os divisores comuns. Note-se que tomando d_1 e d_2, ambos mdc dos mesmos números, então pela definição acima

$$d_1 \leq d_2 \quad e \quad d_2 \leq d_1 \Rightarrow d_1 = d_2.$$

Isto significa que quando o mdc de dois números existe, ele é único (*unicidade do mdc*). Para ser mais "exato", ele sempre existe. Isso é garantido, pelo seguinte lema:

Lema 1 (Lema de Euclides) *Sejam a, b, $n \in \mathbb{N}$, onde $a < na < b$. Se $(a, b - na)$ existe, então (a, b) também existe, e ambos tem o mesmo valor.*

Demonstração:
Chamemos de $d = (a, b - na)$. Logo, $d|a$ e $d|(b - na)$, e pela **Proposição 2.1.3**, temos que $d|(b - na + na) \Rightarrow d|b$. Assim, d divide a e b, restando mostrar que é o máximo divisor. Seja um d' divisor comum a a e b. Também pela propriedade supracitada, $d'|(b - na)$, e desta forma, $d'|d \Rightarrow d' \leq d$. Logo, $d = (a, b)$.

■

Definição 4 *Sejam a e b dois inteiros não simultaneamente nulos. Dizemos que eles são coprimos, ou primos entre si, se e somente se o $mdc(a, b) = 1$.*

Existe uma importante implicação sobre essa definição, apresentada abaixo.

Teorema 2.2.3 *Dois inteiros a e b são coprimos se e somente se existem inteiros x e y tais que $ax + by = 1$.*

Demonstração:
($\Rightarrow$) Se a e b são coprimos, então temos que

$$\text{mdc}(a, b) = 1 \Rightarrow ax + by = 1$$

com x e y inteiros.
($\Leftarrow$) Suponha $d = \text{mdc}(a, b)$, então $d|a$ e $d|b$, o que, pela **Proposição 2.1.3**, indica que d divide qualquer combinação linear entre a e b, inclusive $d|(ax + by)$. Mas como a e b são coprimos, então existem x e y tais que

$$ax + by = 1 \Rightarrow d|1 \Rightarrow d = \text{mdc}(a, b) = 1.$$

■

Os próximos resultados decorrem diretamente do teorema acima:

Corolário 2.2.2 *Sejam a, $b \in \mathbb{Z}$, não mutuamente nulos. Então*

$$\left(\frac{a}{(a,b)}, \frac{b}{(a,b)}\right) = 1.$$

Demonstração:
Cada uma das frações são inteiros, pois $(a, b)|a$ e $(a, b)|b$. Assim, existem x e y inteiros tais que

$$ax + by = (a, b)$$

logo, dividindo ambos os membros por (a, b):

$$\begin{aligned} \frac{ax}{(a,b)} + \frac{by}{(a,b)} &= \frac{(a,b)}{(a,b)} \\ \frac{a}{(a,b)}x + \frac{b}{(a,b)}y &= 1. \end{aligned}$$

■

Corolário 2.2.3 *Sejam a e b coprimos. Então*

$$a|c \quad e \quad b|c \quad \Rightarrow \quad ab|c.$$

Demonstração:
Sejam q, $k \in \mathbb{Z}$ tais que

$$\begin{aligned} a|c &\Rightarrow c = aq; e \\ b|c &\Rightarrow c = bk. \end{aligned}$$

Como $(a, b) = 1$, então existem x, $y \in \mathbb{Z}$ tais que

$$\begin{aligned} &\Rightarrow ax + by = 1 \\ &\Rightarrow axc + byc = c \\ &\Rightarrow ax(bk) + by(aq) = c \\ &\Rightarrow ab(xk + yq) = c \\ &\Rightarrow ab|c. \end{aligned}$$

■

2.3 CONGRUÊNCIAS

Definição 5 *Sejam $a, b \in \mathbb{Z}$ e um $m \in \mathbb{N}$. Dizemos que a e b são* **congruentes módulo m** *se e somente se os restos da divisão euclidiana deles por m são iguais. Quando a é congruente a b módulo m, escrevemos*

$$a \equiv b \bmod m.$$

Em outras palavras, se a congruência não for verificada, dizemos que a e b são *não congruentes* ou *incongruentes*, módulo m. Expressamos $a \not\equiv b \bmod m$.

É importante mencionar que o resto da divisão de qualquer inteiro por 1 é 0, portanto

$$a \equiv b \bmod 1 \quad , \forall a, b \in \mathbb{Z}.$$

Por esse motivo, desconsidere-se este caso; neste trabalho, $m > 1$.

As propriedades abaixo são consequências diretas da definição acima.

Proposição 2.3.1 *Sejam $a, b, c, m \in \mathbb{Z}$, com $m > 1$. Tem-se que*

i) $a \equiv a \bmod m$, *(reflexiva)*

ii) se $a \equiv b \bmod m$, *então* $b \equiv a \bmod m$, *(simétrica)*

iii) se $a \equiv b \bmod m$ *e* $b \equiv c \bmod m$, *então* $a \equiv c \bmod m$. *(transitiva)*

Percebe-se que a congruência módulo m é, então, uma relação de *equivalência* em $\mathbb{Z}$. Além disso, é possível verificar se dois números são congruentes módulo m, sem dividir ambos por m. É suficiente dividir a diferença entre eles ($a - b$ ou $b - a$, já que tratamos de $\mathbb{Z}$). Isso está explícito na propriedade abaixo.

Proposição 2.3.2 *Sejam $a, b, m \in \mathbb{Z}$, com $m > 1$. Então*

$$a \equiv b \bmod m \Leftrightarrow m|(b-a).$$

Demonstração:
($\Rightarrow$) Consideremos a divisão euclidiana de a e b por m:

$$\left.\begin{array}{c} a = mq + r,\ 0 \leq r < m \\ b = mq' + r',\ 0 \leq r' < m \end{array}\right\} \Rightarrow$$

$$\Rightarrow b - a = mq' + r' - mq - r = m(q' - q) + (r' - r).$$

Por hipótese, a e b são congruentes, logo os seus restos na divisão por m são iguais, e portanto

$$r = r' \Rightarrow b - a = m(q' - q) \Rightarrow m|(b-a).$$

($\Leftarrow$) Como $m|b-a$, então existe um $k \in \mathbb{Z}$ tal que $b - a = km \Rightarrow b = km + a$. Dividindo a por m, temos $a = mq + r$. Assim

$$b = km + qm + r = (k+q)m + r.$$

Percebe-se que tanto a quanto b deixam o mesmo resto r na divisão por m, logo

$$\Rightarrow a \equiv b \bmod m.$$

■

As propriedades abaixo mostram as operações de adição e multiplicação usando congruências. Esse teorema é importante para o algoritmo do cálculo de uma data.

Teorema 2.3.1 *Sejam $a, b, c, d, m \in \mathbb{Z}$, com $m > 1$. Então:*

i) se $a \equiv b \bmod m$ e $c > 0$, então $ac \equiv bc \bmod mc$;

ii) se $a \equiv b \bmod m$ e $c \equiv d \bmod m$, então $a + c \equiv b + d \bmod m$ e $ac \equiv bd \bmod m$;

iii) se $a \equiv b \bmod m$, então $a + c \equiv b + c \bmod m$ e $ac \equiv bc \bmod m$.

Demonstração:
i) Como $a \equiv b \bmod m$, então $\exists q \in \mathbb{Z}$ tal que

$$a - b = qm \Rightarrow ac - bc = qmc \Rightarrow ac \equiv bc \bmod mc.$$

ii) Como $a \equiv b \bmod m$ e $c \equiv d \bmod m$, então $\exists q, k \in \mathbb{Z}$ tal que

$$a - b = qm \quad \text{e} \quad c - d = km.$$

Somando membro a membro

$$(a-b) + (c-d) = qm + km \Rightarrow (a+c) - (b+d) = (q+k)m \Rightarrow a + c \equiv b + d \bmod m.$$

Da mesma forma, tomando $a = qm + b$ e $c = km + d$, temos

$$ac - bd = (qm + b)(km + d) - bd = (qkm + qd + bk)m \Rightarrow ac \equiv bd \bmod m.$$

iii) Análoga a *ii)*.

■

Por fim, vê-se que existe o cancelamento aditivo para congruências, mas vejamos como funciona o cancelamento multiplicativo:

Proposição 2.3.3 *Dados a, b, c, $m \in \mathbb{Z}$, com $m > 1$, então*

$$ac \equiv bc \bmod m \Leftrightarrow a \equiv b \bmod \frac{m}{(c, m)}.$$

Demonstração:
Temos que

$$ac \equiv bc \bmod m \Leftrightarrow m|(b - a)c \Leftrightarrow \frac{m}{(c, m)}|(b - a)\frac{c}{(c, m)}.$$

Como, pelo **Teorema 2.2.3**, $\frac{c}{(c, m)}$ e $\frac{m}{(c, m)}$ são coprimos, então

$$\Leftrightarrow \frac{m}{(c, m)}|(b - a) \Leftrightarrow a \equiv b \bmod \frac{m}{(c, m)}.$$

■

2.4 CONGRUÊNCIAS LINEARES E O TEOREMA CHINÊS DOS RESTOS

Se uma congruência do tipo

$$aX \equiv b \bmod m, \quad \text{onde } a, b, m \in \mathbb{Z}, \text{ com } m \geq 1$$

tem solução, significa que procuramos inteiros x tais que $m|(ax - b)$. Usaremos a seguinte proposição:

Proposição 2.4.1 *Dados a, b, $m \in \mathbb{Z}$, com $m \geq 1$, então*

$$\exists x \in \mathbb{Z},\ ax \equiv b \bmod m \Leftrightarrow (a, m)|b.$$

Demonstração:
($\Rightarrow$) Seja $d = (a, m)$. Suponhamos que existe x inteiro tal que $ax \equiv b \bmod m$. Então existe um k tal que

$$ax - b = mk \quad \text{ou} \quad ax - mk = b.$$

Mas $d|a$ e $d|m$, logo, pela **Proposição 2.1.3**, $d|(ax - mk) \Rightarrow d|b$.

($\Leftarrow$) Suponhamos agora que $d|b$, então existe um $q \in \mathbb{Z}$ tal que $b = dq$. Ainda, tomemos x e y tais que $ax - my = d$. Temos:

$$\begin{aligned} ax - my &= d \\ \Leftrightarrow\ aqx - mqy &= dq = b \\ \Leftrightarrow\ aqx - b &= mqy \\ \Leftrightarrow\ aqx &\equiv b \bmod m \end{aligned}$$

com qx sendo uma solução da congruência linear.

■

Dizemos que, dado um x_i encontrado como solução da congruência

$$aX \equiv b \bmod m$$

este é uma *solução particular*. Isso porque todo x_j, com $i \neq j$, tal que $x_i \equiv x_j \bmod m$, também é uma solução da congruência. Então precisamos saber como encontrar todas as *soluções incongruentes módulo m*. O teorema abaixo garante isso, mas por não estar diretamente relacionado ao objeto do trabalho, sua demonstração será omitida, podendo ser encontrada em [1] ou [7].

Teorema 2.4.1 *Seja $d = (a, m)$, onde $d|b$. Então a congruência linear*

$$aX \equiv b \bmod m$$

tem exatamente d soluções da forma

$$x_0,\ x_1 = x_0 + \frac{m}{d},\ x_2 = x_0 + \frac{2m}{d}, \ldots,\ x_{d-1} = x_0 + \frac{(d-1)m}{d}$$

tais que são incongruentes duas a duas módulo m.

Congruências lineares resolvem importantes problemas teóricos e práticos (como no **Capítulo 4**). Vamos apresentar o *Teorema Chinês dos Restos*, e um exemplo de sua aplicação.

Teorema 2.4.2 (Teorema Chinês dos Restos) *Sejam m_1, m_2, ..., m_n inteiros positivos coprimos dois a dois. O sistema de congruências lineares*

$$\begin{cases} X \equiv r_1\ mod\, m_1 \\ X \equiv r_2\ mod\, m_2 \\ \quad\ \vdots \\ X \equiv r_n\ mod\, m_n \end{cases}$$

possui uma única solução módulo $M = m_1 m_2 \cdots m_n$.

Demonstração:
Tomemos os números $M_i = \dfrac{M}{m_i}$, com $i = 1, 2, \ldots, n$. Logo, pelo fato dos m_i serem coprimos e pelo **Teorema 2.4.1**, temos que

$$(M_i, m_i) = 1 | r_i \Rightarrow \exists x_i \text{ tal que } M_i x_i \equiv 1 \bmod m_i. \tag{2.1}$$

Com isso, uma solução do sistema é

$$x = M_1 r_1 x_1 + M_2 r_2 x_2 + \ldots + M_n r_n x_n.$$

De fato, $m_i | M_j$, com $i \neq j$, logo

$$M_j \equiv 0 \bmod m_i \Rightarrow \begin{cases} M_1 r_1 x_1 & \equiv & 0 \bmod m_i \\ M_2 r_2 x_2 & \equiv & 0 \bmod m_i \\ & \vdots & \\ M_i r_i x_i & \equiv & M_i r_i x_i \bmod m_i \\ & \vdots & \\ M_n r_n x_n & \equiv & 0 \bmod m_i \end{cases}$$

Somando as congruências, temos $x \equiv M_i r_i x_i \bmod m_i$. Sabendo de (2.1), então

$$X \equiv r_i \cdot 1 \equiv r_i \bmod m_i.$$

A solução existe, mostremos então que é única. Suponhamos y outra solução do sistema, logo

$$y \equiv x \bmod m_i \Rightarrow m_i | (y - x).$$

Como os m_i são coprimos, diretamente pelo **Corolário 2.2.3**, temos que:

$$M = m_1 m_2 \cdots m_n | (y - x) \Rightarrow y \equiv x \bmod M.$$

Logo o sistema possui única solução módulo M.

■

Considerando o exposto, temos que $M | (x - \sum_{i=1}^{n} M_i r_i x_i)$, ou seja

$$x = \sum_{i=1}^{n} M_i r_i x_i + Mk, \quad \text{com } k \in \mathbb{Z}.$$

Chamaremos a expressão acima de *solução geral* do sistema de equações lineares. Apliquemos num exemplo.

Exemplo 5 ([15]) Vamos encontrar um inteiro x tal que

$$\left.\begin{array}{rcl} 5X & \equiv & 1 \bmod 19 \\ 5X & \equiv & 16 \bmod 18 \end{array}\right\} \tag{2.2}$$

Primeiramente, temos que:

$$\left.\begin{array}{rcrcrcl} 5X & \equiv & 1 & \equiv & 1+19 & \equiv & 20 \mod 19 \\ 5X & \equiv & 16 & \equiv & 16+3\cdot 18 & \equiv & 70 \mod 18 \end{array}\right\}$$

Como $(5,19) = (5,18) = 1$, podemos usar o resultado da **Proposição 2.3.3**:

$$\left.\begin{array}{rcl} X & \equiv & 4 \mod 19 \\ X & \equiv & 14 \mod 18 \end{array}\right\} \tag{2.3}$$

Significa que resolver o sistema (2.3) é equivalente a resolver o sistema (2.2). Aplicando o Teorema Chinês dos Restos, temos

$$M = 19 \cdot 18 = 342,\ M_1 = 18,\ M_2 = 19.$$

Temos também que uma de suas soluções é

$$x \equiv 18 \cdot 4x_1 + 19 \cdot 14x_2 \mod 342. \tag{2.4}$$

Encontremos x_1 e x_2:

$$\left.\begin{array}{rcrcl} 18x_1 & \equiv & 1 \mod 19 & \Rightarrow & x_1 = 18 \\ 19x_2 & \equiv & 1 \mod 18 & \Rightarrow & x_2 = 1 \end{array}\right\} \tag{2.5}$$

Substituindo (2.5) em (2.4), temos

$$x \equiv 72 \cdot 18 + 266 \cdot 1 \equiv 1562 \equiv 194 \mod 342.$$

Vejamos que $5 \cdot 194 = 970 = 19 \cdot 51 + 1 = 18 \cdot 53 + 16$.

$\diamond$

3 Linguagem C++: Noções Básicas

A princípio, é bom diferenciar algoritmo de programa. Para o primeiro, Davis declara em [5] que "é uma descrição das etapas a serem executadas, normalmente a um alto nível de abstração. Um algoritmo é detalhado, mas geral". Pode ser usada qualquer linguagem que consiga descrever a sequência de passos e os objetos tratados. Em contrapartida, um programa é o resultado de um algoritmo traduzido para uma linguagem de programação, para, através de um *compilador*[1], efetuar as tarefas às quais foi destinado.

Um algoritmo como o da divisão euclidiana, para encontrar os inteiros q e r tais que na divisão de a por b tenhamos $a = bq + r$, ou o chamado algoritmo de Euclides para encontrar o *máximo divisor comum* entre dois números, são bons exemplos de "sequências de passo a passo". O programa (=software) num *smartphone* que reconhece o número digitado como sendo da agenda e efetua uma ligação para o mesmo, apesar de ter uma "sequência de tarefas, com entrada e saída de valores", é executado por um sistema operacional, portanto, não é apenas um algoritmo.

A linguagem de programação escolhida para este trabalho é o C++, linguagem derivada do antigo (e ainda usado) C, porém mais aprimorado. O programa utilizado como compilador e editor de texto[2] para a elaboração das rotinas aqui presentes é o Code::Blocks[3], podendo ser usado qualquer um, como o Dev-C++[4], que assim como o anterior, é gratuito.

É bom salientar neste ponto que alguns elementos necessários a um completo entendimento da programação, como o funcionamento de um compilador, o conceito de computador simplificado, definição de estruturas e classes, entre outros, serão omitidos, por divergirem do objetivo deste texto. Para maior aprofundamento, ver [3], [5] ou [10].

3.1 INICIANDO UM PRIMEIRO PROGRAMA

Antes de tudo, alguns detalhes. A linguagem C++ é *case sensitive*, o que significa que diferencia maiúsculas de minúsculas. A quantidade de espaços também não influencia no programa, desde que não estejam no meio do nome de palavras chaves, comandos e variáveis. Estas podem iniciar seu nome apenas com uma letra ou um caractere de sublinhado, e no resto do nome pode-se usar algarismos também. Além disso,

[1] Software que converte a linguagem de programação, chamada de alto nível, para a linguagem de máquina (binária), chamada de baixo nível.

[2] Qualquer programa que edite texto serve, porém é desvantajoso usar um editor qualquer, pois um apropriado ao C++ pode analisar e apontar erros, completar comandos e até contribuir com um melhor design gráfico, separando comandos, variáveis, texto, comentários, etc.

[3] http://www.codeblocks.org/. Versões disponíveis para Windows, Macintosh e Linux.

[4] http://www.bloodshed.net/devcpp.html. Versões disponíveis para Windows e Linux/Unix.

é necessário um certo "cuidado" com acentos, na hora de exibir textos. Nos programas abaixo, estes serão evitados.

Um programa o mais simples possível precisa de alguns elementos básicos essenciais para poder ser compilado[5]. Ficaria assim:

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5   return 0;
6 }
```

As instruções em C++ sempre terminam com ponto e vírgula, porém vemos que na primeira linha deste primeiro programa não tem. É o que chamamos de *diretiva do pré-processador*. Este é um programa que analisa o programa-fonte e executa algumas modificações conforme as diretivas, que são funções (instruções) específicas para ele, que não fazem parte da linguagem C++. Sempre começam com #. Esta diretiva (**#include**) inclui outro arquivo exatamente no local onde é *chamada*, copiando o seu conteúdo e colando neste ponto. Neste caso, o arquivo é a biblioteca[6] **iostream**, responsável pelo gerenciamento de *entrada* e *saída* de dados no programa.

O comando **using** serve para dizer ao compilador onde estão (*espaço de trabalho*) as funções e classes presentes no programa, e neste caso, no **namespace std** (=*standard*, padrão). Neste "namespace " estão, por exemplo, os comandos **cin**>> e **cout**<<, que tão usados serão neste trabalho.

A função **main** (=*principal*) é "a função" do programa, isto é, o que queremos fazer estará dentro de **main**; começa após o { e termina no }. Vê-se que a única instrução da linguagem C++ mesmo neste primeiro programa é o "return 0", que significa retornar valor zero, já que a função **main** é do tipo inteiro, logo deve devolver "a quem a chamou" um inteiro, e neste exemplo como não deveria fazer coisa alguma, retorna o valor nulo. Nos parênteses devem ser colocados certos parâmetros, de acordo com o uso específico de **main**. Para uso neste trabalho, os parênteses estarão sempre vazios.

Para não ficarmos apenas com este primeiro programa "trivial", tomaremos mais quatro instruções, comuns no uso do C++: **cin**, **cout**, declaração de variável e atribuição (=).

Exemplo 6 Vejamos como criar um programa que peça 4 números naturais ao usuário e exiba na tela a média aritmética desse conjunto. Neste caso, deverá aparecer uma mensagem na tela pedindo cada número por vez, e para isso usaremos o comando **cout**

[5] Doravante, entenda como montado, traduzido em linguagem de máquina.

[6] Biblioteca é um conjunto de comandos e funções elaborados para situações específicas dentro de um programa. Por exemplo, a biblioteca **cstdlib**, contém um comando chamado **rand**(), que gera um número pseudo-aleatório.

(*saída*), com o **operador de inserção**[7] <<. Para ler cada valor e armazená-lo, usaremos **cin** (*entrada*), acompanhado do **operador de extração**[8] >>. Mas para armazenar, devemos declarar *variáveis*. Uma variável é um espaço de memória destinada a armazenar um tipo de dado. Toda variável deve ser declarada (e muitas vezes *inicializada*) antes do seu uso. Chamaremos de `a`, `b`, `c` e `d`, além de uma `m`, onde será guardada o valor da média. Para declará-las, primeiro diz-se seu tipo, podendo ser basicamente *inteira* (**int**), *real* (**float** ou **double** – decimais), *lógica* (**bool** – verdadeiro ou falso), *caractere* (**char** – um caractere, imprimível ou não) e *void* (**void** – o "nada"). O nosso programa ficaria assim:

```
1  #include <iostream>   // Isso eh um comentario, sempre apos as barras
2  using namespace std;  // duplas. O compilador ignora essa parte do
3                        // programa.
4  int main()
5  {
6     float a,b,c,d,m;  // Perceba que todas as variaveis foram declaradas
7                       // juntas (sao todas do mesmo tipo), mas nao foram
8                       // inicializadas
9     cout<<"Digite o primeiro numero: ";
10    cin>>a;
11    cout<<"\nDigite o segundo numero: ";
12    cin>>b;
13    cout<<"\nDigite o terceiro numero: ";
14    cin>>c;
15    cout<<"\nDigite o quarto numero: ";
16    cin>>d;
17    m=(a+b+c+d)/4;
18    cout<<"\n\nMedia dos quatro numeros digitados: "<<m;
19    return 0;
20 }
```

No início do programa, no primeiro pedido, espera-se a interação do usuário. O mesmo digita um número, e ao teclar ENTER, o programa registra esse valor na variável `a`, ao que em seguida o programa já faz o próximo pedido. Após o registro na variável `d`, o programa calcula a média dos 4 números e *guarda* o valor na variável `m`. Assim funciona o *operador de atribuição* =. Ele copia o valor que está (ou que resulta, no caso de expressões) do lado *direito* e registra na variável que está do lado *esquerdo*. Note-se que as variáveis declaradas são todas do tipo *float*[9], pelo fato da média poder precisar de casas decimais para ser expressa. Em seguida, o programa exibe a **string**[10] "`Media dos quatro numeros digitados: `" e o valor de `m`, previamente calculado. O programa, ao

[7] Serve para inserir dados na **stream** que será exibida. **Stream** é uma sequência de dados.
[8] Para extrair os dados da **stream** inserida.
[9] O tipo *double*, que também armazena números decimais, reserva uma quantidade maior de memória, por isso a escolha por *float*.
[10] Sequência de caracteres; um tipo de vetor, que funciona como variável.

final, exibe o tempo de execução, e aguarda o usuário teclar algum botão para fechar (ou continuar, quando for o caso). O caractere "\n" passa para a próxima linha. Um exemplo da tela do executável[11] segue abaixo.

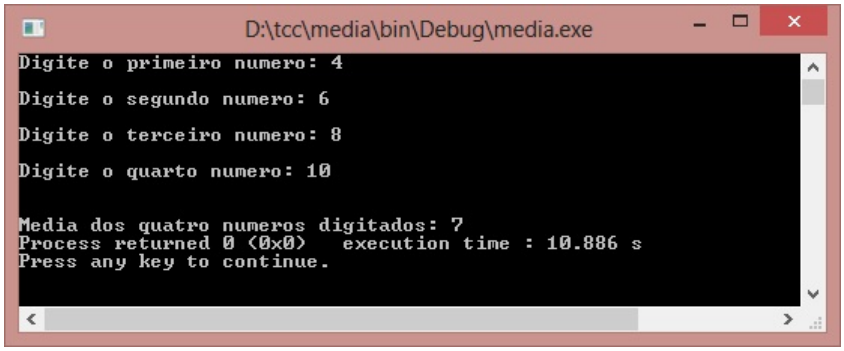

Figura 1 – Executável do programa **media**
Fonte: própria.

◇

3.2 REPETINDO PASSOS: LAÇOS

Em diversas situações é necessário que uma certa instrução (ou instruções) sejam efetuadas algumas vezes, até que se atinja um resultado esperado, ou uma quantidade satisfatória. Para isso, nas linguagens de programação, se usa o *laço* (do inglês *loop*, como normalmente é conhecido). Por exemplo, para se calcular o quociente e o resto de no algoritmo da divisão, pode ser um procedimento relativamente lento, e a velocidade reduz se a diferença entre divisor e dividendo aumenta, pois é preciso que se encontre um múltiplo do divisor o maior possível, mas que seja menor ou igual ao dividendo (ver **Exemplo 8**). Porém, aproveitando a praticidade de um programa em C++, facilmente se encontra quociente e resto de uma divisão desta forma. Mas antes, serão apresentados os tipos de laços usados no C++.

O laço **for** geralmente é usado quando se quer repetir uma instrução uma quantidade de vezes fixa, determinada.

```
1 for(inicializacao; teste; incremento)
```

Uma variável é escolhida para funcionar como *contador*; a ele se referem os três termos no parênteses. Após a chamada do laço, ele é inicializado (podendo até ser declarado nesse ponto). Em seguida faz-se o teste (no início do laço), lembrando que por se tratar de um contador, geralmente é uma comparação de ordem ($>$, $\geq$, $<$ ou $\leq$). Esse teste é feito toda vez que o laço é reiniciado. Se o teste resultar em *verdadeiro*, ele cumpre a instrução dada e, no fim do laço, antes de reiniciar, aumenta (ou diminui) o valor do

[11] Programa que realmente "executa" as instruções determinadas, gerado pelo compilador.

contador. Recomeça, refazendo o teste. O laço **for** não exige ponto-e-vírgula, porém a instrução ligada ao laço, sim. Caso haja mais de uma instrução a repetir, só agrupá-las entre chaves.

Exemplo 7 Vamos ver como escrever os 30 primeiros termos da sequência de Fibonacci. Considerando $F_1 = F_2 = 1$ os dois primeiros termos da sequência, e sabendo então que $F_{n+2} = F_n + F_{n+1}$, não é nada difícil calcular tais termos, porém, a quantidade exige um certo esforço. Por se tratar de uma recorrência, o mesmo cálculo é repetido algumas vezes para encontrar o termo seguinte. Por esse motivo, o laço se torna interessante. O programa ficaria o seguinte:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int i,fn2=2,fn1=1,fn,fax;
6      cout<<1<<"\t"<<1<<"\t";
7      for(i=0;i<28;i++)
8      {
9          cout<<fn2<<"\t";
10         fn=fn1;
11         fax=fn2;
12         fn2=fn2+fn1;
13         fn1=fax;
14     }
15     return 0;
16 }
```

Foram usadas 5 variáveis nesse programa: três variáveis para a sequência (`fn2`, `fn1` e `fn`, para F_{n+2}, F_{n+1} e F_n, respectivamente), uma para a contagem do laço (`i`) e outra auxiliar (`fax`). Foram escritos os dois primeiros termos da sequência, e justamente por isso, o teste só conta até `i`< 28 no laço. Dentro deste, ele imprime `fn2`$= 2$ (declarada e já inicializada) com o comando **cout**<<. Neste ponto, os valores são

`fn2`$= 2$ `fax` (sem valor) `fn1`$= 1$ `fn` (sem valor)

Daí começa os "repasses" de valores. É preciso destacar que as quatro linhas restantes do laço não são igualdades, são atribuições (valor do lado direito *atribuído* à variável do lado esquerdo). Assim, `fn` recebe o valor de `fn1`, `fax` recebe o de `fn2` (e reserva!), `fn2` recebe o próprio valor somado ao que está em `fn1`, e por fim, `fn1` recebe o que estava reservado em `fax`. Este ciclo se repete 28 vezes, já que `i` começa com valor nulo. A imagem do programa em execução seria assim:

No fim, apenas `fn2` estava sendo exibido, mas sempre com um valor diferente. O caractere "`\t`" imprime o espaço de um parágrafo.

◇

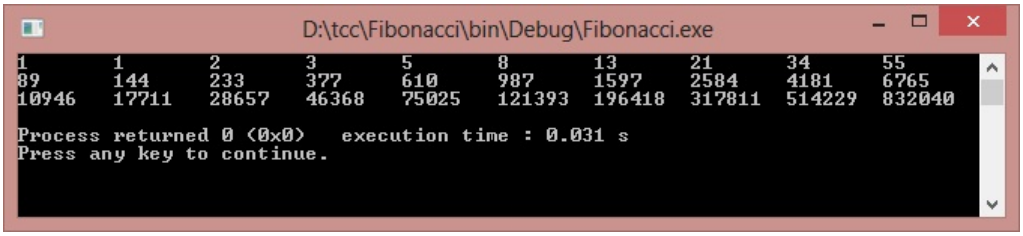

Figura 2 – Executável do programa **Fibonacci**
Fonte: própria.

O segundo laço é o **while**. Um pouco parecido com o **for**, mas a condição de término pode não ser fixa, isto é, o laço pode encerrar de acordo com um fator externo. Como no **for**, ele pode ser usado com um contador (para ser testado), ou um teste de igualdade, e este é efetuado antes de começar; sendo verdadeiro, efetua as instruções correspondentes.

Exemplo 8 Assim seria um programa que calcula o quociente e o resto da divisão de 476 por 25, usando o algoritmo da divisão. Neste caso, serão testados todos os múltiplos de 25, até encontrar o maior desses múltiplos que seja menor ou igual a 476. O programa ficaria assim:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int q=0,r;
6      while((q*25)<=476)
7      {
8          cout<<q<<" * 25 = "<<q*25<<"\t";
9          q++;
10     }
11     q--;
12     r=476-q*25;
13     cout<<"\n\nDividindo 476 por 25 obtemos:\n";
14     cout<<"quociente= "<<q<<endl;
15     cout<<"resto= "<<r<<endl;
16     return 0;
17 }
```

A princípio, o quociente é inicializado (`q=0`) antes do laço, e seu produto por 25 é testado, comparado ao dividendo (476). A cada iteração do laço, é impresso o produto de `q` por 25, e o mesmo é incrementado em 1 unidade (`q++`, que é igual a atribuição `q=q+1`). Quando `q` aumenta a ponto do produto ser maior que 476, o laço não inicia, e o programa segue na próxima linha. Como `q` aumentou mais do que devia, retorna-se assim ao valor anterior, subtraindo 1 (`q--`). Calcula-se então o resto com o quociente correto, e enfim imprime-se o resultado. O comando `endl` pula para a próxima linha, como o

caractere "\n". Uma imagem do executável encontra-se abaixo.

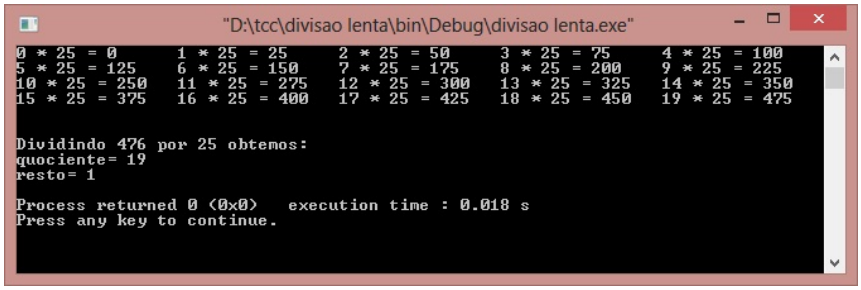

Figura 3 – Executável do programa **divisao lenta**
Fonte: própria.

A verdade é que esse programa poderia ter calculado mais rápido, sem usar o **while**, mas usando dois detalhes da linguagem C++. Ficaria assim:

```
#include <iostream>
using namespace std;
int main()
{
    int q,r;
    q=476/25;
    r=476%25;
    cout<<"Dividindo 476 por 25 obtemos:\n";
    cout<<"quociente= "<<q<<endl;
    cout<<"resto= "<<r<<endl;
    return 0;
}
```

Primeiro, as duas variáveis são inteiras (`q` e `r`), mas como o resto é diferente de 0, significa que a divisão não é exata, ou seja, o quociente 476/25 tem casas decimais. Em C++, um número inteiro que recebe um valor decimal só registra a parte inteira desse número. Isto tudo garante que `q` recebe o quociente da divisão. Em segundo lugar, o símbolo de porcentagem é usado para calcular o resto da divisão, ou seja, podemos usá-lo para calcular módulo. O que foi usado neste programa alternativo será muito usado no **Capítulo 4**.

◇

Para encerrar esta seção, o terceiro tipo de laço é o **do-while**. Ele é muito semelhante ao **while**. A diferença é que o programa sempre inicia o laço, realizando o teste após cada iteração. Sua sintaxe ficaria:

```
do
{
   instrucoes;
   ...
}while(teste);
```

O detalhe aqui é o uso de ponto-e-vírgula após os parênteses do teste.

3.3 TOMANDO DECISÕES: IF E ELSE

O objetivo de se programar é fazer com que o computador realize tarefas (na maioria das vezes dispendiosas), e pra isso às vezes é preciso tomar algumas decisões. Isso significa que o programa deve saber o que fazer caso determinadas *condições* sejam atendidas; caso contrário, não realiza aquela(s) instrução(ões) específica(s). Para incrementar ainda mais, é possível fazê-lo cumprir outros comandos somente na falta da condição determinada. Estamos falando dos comandos **if** e **else**.

O comando **if** (*se*, em inglês) tem sintaxe bem simples. Após chamá-lo no programa, é colocado uma condição entre parênteses, e a instrução seguinte só é efetuada caso o teste retorne em *verdadeiro*, senão simplesmente a pula. Caso haja mais de uma instrução (*condicional*), só agrupá-las entre chaves.

Exemplo 9 Apresentamos a seguir um programa que conta quantos múltiplos de um certo inteiro a existem entre dois outros inteiros b e c, tais que $1 < a < b < c$. Aqui, o programa deve perguntar ao usuário valores para registrar nas três variáveis da questão. Em seguida procurar quantos múltiplos de a estão entre b e c, sendo $b < ma < c, \forall m \in \mathbb{N}$, com $m > 1$. O programa ficaria assim:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a,b,c,m=2,n=0;
6      cout<<"Digite um valor para a: ";
7      cin>>a;
8      cout<<"\nDigite um valor para b: ";
9      cin>>b;
10     cout<<"\nDigite um valor para c: ";
11     cin>>c;
12     while((m*a)<c)
13     {
14         if(m*a>b) n++;
15         m++;
16     }
17     cout<<"\n\nExistem "<<n<<" multiplos de "<<a<<" entre "<<b<<" e "<<c
           ;
18     return 0;
19 }
```

Após o registro dos três valores, o programa testa, no **while**, se o dobro de `a` é menor que `c`, haja vista que `1a=a<b`, logo estaria trivialmente fora das condições; por isso `m` inicializa com 2. E se `2a>c`, o programa nem entra no laço. Supondo `2a<c`, a primeira instrução agora testa *se* também é maior que `b`. Se for, acrescenta 1 ao contador de múltiplos `n`. Cumprindo ou não a condição do **if**, o programa deve testar o próximo

múltiplo, por isso `m` incrementa em 1 sempre que entra no laço. Após isso, o laço repete tantas vezes forem necessárias para até `ma` ultrapassar `c`. No fim, é exibida a quantidade de múltiplos de `a` entre `b` e `c`. A seguir, uma imagem do executável.

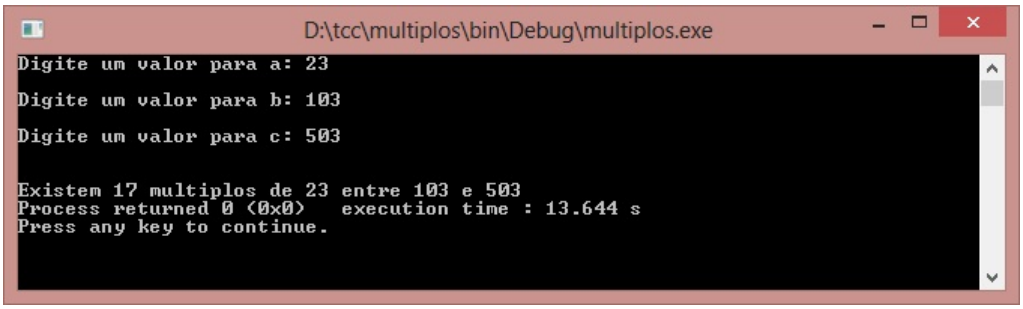

Figura 4 – Executável do programa **multiplos**
Fonte: própria.

Uma alternativa a este programa, que o tornaria mais completo, é se usássemos o comando **else** (*senão*, em inglês). Isto significa que é possível fazer com que o programa realize certas instruções somente se o teste do **if** for verdadeiro, e outras determinadas funções somente se esse teste for falso. Implementando este comando no programa anterior, teríamos:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5    int a,b,c,m=2,n=0;
6    cout<<"Digite um valor para a: ";
7    cin>>a;
8    cout<<"\nDigite um valor para b: ";
9    cin>>b;
10   cout<<"\nDigite um valor para c: ";
11   cin>>c;
12   if(2*a<c)
13   {
14     while((m*a)<c)
15     {
16       if(m*a>b) n++;
17       m++;
18     }
19     if(n==0)
20       cout<<"\n\nNao existem multiplos de "<<a<<" entre "<<b<<" e "<<c;
21     else
22       cout<<"\n\nExistem "<<n<<" multiplos de "<<a<<" entre "<<b<<" e "
             <<c;
23   }
24   else
25   {
```

```
26      cout<<"\n\nNao existem multiplos de "<<a<<" entre "<<b<<" e "<<c;
27    }
28    return 0;
29  }
```

Veja que neste caso, temos um teste logo no início: o programa só testará o laço, se 2a for menor que c. Após a contagem no laço, o programa compara n com 0 (o operador == significa "igual"), e se for igual, assim como se nem entrar no laço, exibirá que "não existe multiplos de a entre b e c". Se não for n==0, é porque tem múltiplos, e então será exibida a outra frase com tal quantidade.

Uma segunda alternativa seria usar o resultado proposto em 2.2.1. O programa ficaria bem mais simples:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a,b,c,n;
6      cout<<"Digite um valor para a: ";
7      cin>>a;
8      cout<<"\nDigite um valor para b: ";
9      cin>>b;
10     cout<<"\nDigite um valor para c: ";
11     cin>>c;
12     n=((c-1)/a)-(b/a);
13     if(n!=0) cout<<"\n\nExistem "<<n<<" multiplos de "<<a<<" entre "<<b
           <<" e "<<c;
14     else cout<<"\n\nNao existem multiplos de "<<a<<" entre "<<b<<" e "<<
           c;
15     return 0;
16 }
```

Para concluir, vemos no teste do **if** um *operador lógico*: **NÃO**, representado por !=, e usado acima como *diferente*, ou *não igual*. Além deste, também é comum o uso dos operadores lógicos **E** e **OU**, representados respectivamente por && e ||.

◇

Um outro comando útil em C++ é o comando **switch**. Ele pode substituir os *condicionais* **if-else** nos casos em que há vários caminhos a serem direcionados. Na sua sintaxe, ele compara a variável (ou expressão) em questão com cada valor *constante* declarado em cada **case** (*caso*, em inglês). Tendo valor igual a constante, o programa então efetua as instruções relativas a esse **case**. O caso **default** é para quando o teste não corresponder a nenhum dos casos declarados. O comando **break** interrompe o **switch**, fazendo com que seja cumprido apenas o que está dentro do caso determinado. Sem ele, o programa efetuará as instruções do caso seguinte, e assim por diante, perdendo o sentido do **switch**.

```
switch(variavel)
{
   case valor1:
      instrucoes;
      ...
      break;
   case valor2:
      instrucoes;
      ...
      break;
   case ...
      instrucoes;
      ...
      break;
   default:
      instrucoes;
      ...
}
```

4 A Páscoa em Linguagem C++

Para início, vamos determinar alguns parâmetros para o início dos nossos cálculos. Tomaremos como base o ano de 1600, pois foi o primeiro ano que "deixou de ser bissexto". Como o dia acrescentado (quando acontece) é 29 de fevereiro, esse será o último dia do ano anterior. Então o primeiro dia do ano é 1º de março, terminando em fevereiro, o 12º mês. Tomemos também um número para cada dia da semana, a partir do 0 para o domingo, até o 6 para o sábado.

4.1 CONGRUÊNCIA DE ZELLER

Primeiramente, encontraremos uma expressão que determina o primeiro dia (1º de março) de um ano Y qualquer. Definiremos aqui z_Y como sendo esse dia, ou seja:

$$1^o/\text{março}/Y \Rightarrow z_Y.$$

Um ano qualquer inicia 365 dias após o ano anterior, isto é, sabendo que $365 \equiv 1 \bmod 7$, um dia da semana depois. No caso do ano anterior ter sido bissexto, ou seja, ter havido o 29 de fevereiro, teremos 2 dias a mais. Então o ano de 1601 começa no dia $z_{1600} + 1 \bmod 7$ (lembrando que estamos usando números de 0 a 6 para representar os dias da semana)[1]. Da mesma forma, o ano 1602 começa no dia $z_{1600} + 2 \bmod 7$, e 1603 começa no dia $z_{1600} + 3 \bmod 7$. Porém 1604 começa após um 29 de fevereiro, logo seu primeiro dia é $z_{1600} + 4 + 1 \bmod 7$. Os anos bissextos interferem no cálculo.

Temos que levar em conta quantos anos bissextos tem entre 1600 e o ano Y. Baseado na **Proposição 2.2.1**, temos que contar a quantidade de múltiplos de 4 maiores que 1600 e menores que ou iguais a Y, e descontar dessa quantidade aqueles que são múltiplos de 100 e não são de 400. No final disso, teremos o total de anos bissextos no intervalo acima. Chamaremos esse número de b. Segue assim:

$$\begin{aligned} b &= \underbrace{\left(\left[\frac{Y}{4}\right] - \left[\frac{1600}{4}\right]\right)}_{\text{múltiplos de 4}} - \left(\underbrace{\left(\left[\frac{Y}{100}\right] - \left[\frac{1600}{100}\right]\right)}_{\text{múltiplos de 100}} - \underbrace{\left(\left[\frac{Y}{400}\right] - \left[\frac{1600}{400}\right]\right)}_{\text{múltiplos de 400}}\right) \\ &= \left[\frac{Y}{4}\right] - 400 - \left(\left[\frac{Y}{100}\right] - 16 - \left[\frac{Y}{400}\right] + 4\right) \\ &= \left[\frac{Y}{4}\right] - \left[\frac{Y}{100}\right] + \left[\frac{Y}{400}\right] - 388. \end{aligned}$$

Com isso, determinamos uma expressão para b:

[1] Essa notação nos dá o resto da divisão por 7.

Proposição 4.1.1 *O número de anos bissextos entre* 1600 *e um certo ano* Y *é dado por*

$$b = \left[\frac{Y}{4}\right] - \left[\frac{Y}{100}\right] + \left[\frac{Y}{400}\right] - 388.$$

Organizemos agora tudo numa única relação:

$$z_Y = (z_{1600} + (Y - 1600) + b) \bmod 7. \tag{4.1}$$

Isso significa que z_Y é dado contando z_{1600} mais um dia a cada ano que se passa de 1600 até o ano Y (nesse caso, $Y-1600$), mais um dia para cada ano bissexto (b). Agora, precisamos de uma referência. Sabendo que o dia 1º de março de 2017 foi numa quarta-feira, temos

$$z_{2017} = 3. \tag{4.2}$$

Calculemos b para este ano:

$$\begin{aligned} b &= \left[\frac{2017}{4}\right] - \left[\frac{2017}{100}\right] + \left[\frac{2017}{400}\right] - 388 \\ b &= 101. \end{aligned} \tag{4.3}$$

Como $101 \equiv 3 \bmod 7$, de (4.2) e (4.3), aplicados em (4.1), temos:

$$\begin{aligned} 3 &= (z_{1600} + 2017 - 1600 + 101) \bmod 7 \\ 3 &= (z_{1600} + 518) \bmod 7. \end{aligned}$$

Sabendo que $518 \equiv 0 \bmod 7$, conseguimos então definir z_{1600}:

$$z_{1600} \equiv 3 \bmod 7.$$

Esse resultado nos indica que o dia da semana de 1º de março de 1600 também foi uma quarta-feira. Podemos agora definir uma expressão para z_Y:

$$z_Y = (3 + Y - 1600 + \left[\frac{Y}{4}\right] - \left[\frac{Y}{100}\right] + \left[\frac{Y}{400}\right] - 388) \bmod 7.$$

Como $1600 + 388 \equiv 0 \bmod 7$, concluimos a nossa expressão:

Proposição 4.1.2 *O primeiro dia de março de um ano* Y *qualquer é dado por*

$$z_Y = \left(3 + Y + \left[\frac{Y}{4}\right] - \left[\frac{Y}{100}\right] + \left[\frac{Y}{400}\right]\right) \ mod 7.$$

Seguindo daqui, devemos calcular agora o primeiro dia de cada mês de um certo ano Y. Aproveitaremos a notação anterior: seja $z_{m,Y}$ o primeiro dia do mês m do ano Y. Temos que do dia 1º de um certo mês para o dia 1º do mês seguinte são somados 30 ou 31 dias. Porém,

$$30 \equiv 2 \bmod 7 \quad \text{e} \quad 31 \equiv 3 \bmod 7$$

então, na prática, são somados 2 ou 3 dias ao dia da semana, assim:

$$\begin{aligned} z_{2,Y} &\equiv (z_{1,Y} + 3) \bmod 7 \\ z_{3,Y} &\equiv (z_{2,Y} + 2) \equiv (z_{1,Y} + 5) \bmod 7 \\ z_{4,Y} &\equiv (z_{3,Y} + 3) \equiv (z_{1,Y} + 8) \bmod 7 \ldots \end{aligned}$$

Resumindo, devemos somar, para cada mês, o valor correspondente na tabela abaixo:

mar	abr	mai	jun	jul	ago	set	out	nov	dez	jan	fev
3	6	8	11	13	16	19	21	24	26	29	32

Tabela 2 – Constantes da congruência de Zeller, para cada mês.
Fonte: própria.

ressaltando que o 3 na coluna de março "se refere" ao mesmo presente na **Proposição 4.1.2**. A seguinte fórmula empírica nos fornece esses valores para cada $m = 1, 2, \ldots, 12$:

$$1 + \left[\frac{13m - 1}{5}\right]. \tag{4.4}$$

Uma forma de mostrar a validade da expressão acima é através do gráfico das funções $f, g : \mathbb{R} \to \mathbb{R}$, tais que $f(x) = 2,6x + 0,8$ e $g(x) = 2,6x - 0,2$:

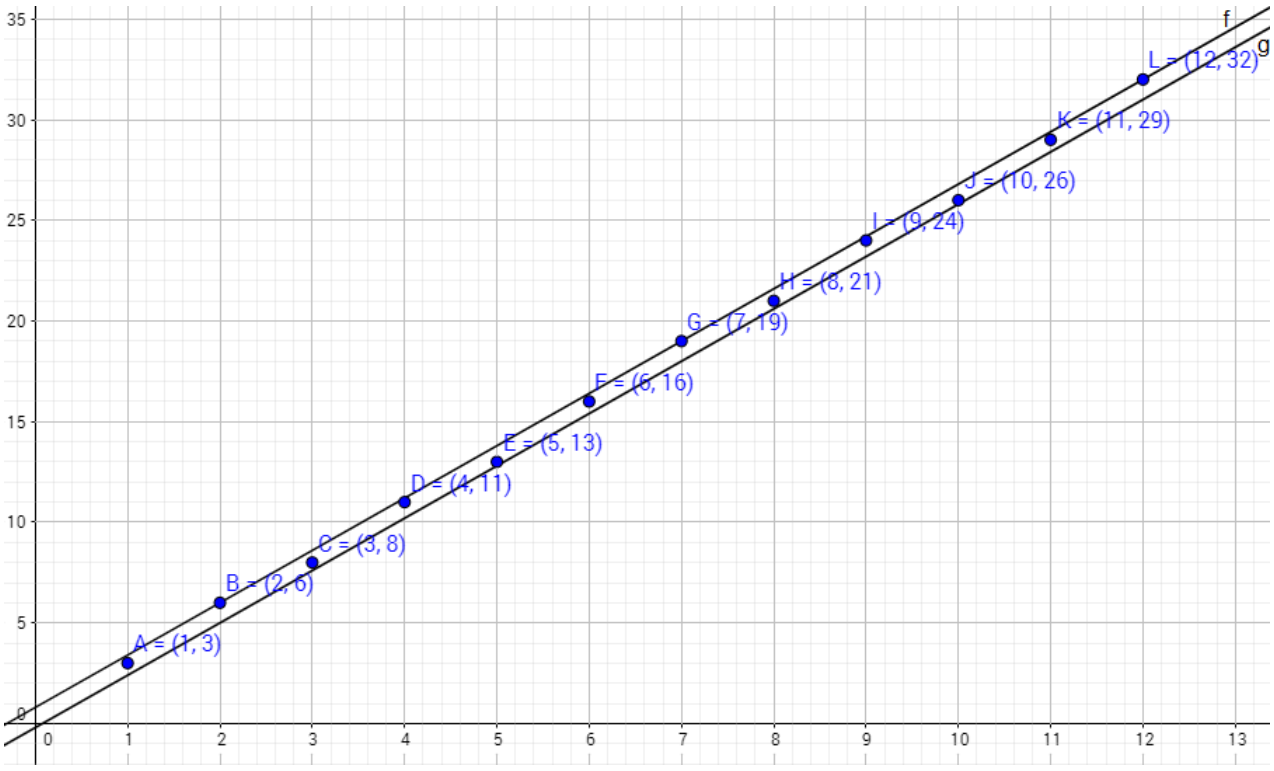

Figura 5 – Gráfico das funções $f(x) = 2,6x + 0,8$ e $g(x) = 2,6x - 0,2$
Fonte: *Geogebra*.

A figura acima nos mostra que os pontos

$$(1, 3), (2, 6), (3, 8), \ldots, (12, 32)$$

que nos interessam (da tabela acima), estarão abaixo da reta do gráfico de f, porém acima do gráfico de g.

Substituindo então 3 por (4.4) na expressão de z_Y, da **Proposição 4.1.2**, temos:

Proposição 4.1.3 *O primeiro dia do mês m de um dado ano Y é dado por*

$$z_{m,Y} = 1 + \left[\frac{13m-1}{5}\right] + Y + \left[\frac{Y}{4}\right] - \left[\frac{Y}{100}\right] + \left[\frac{Y}{400}\right] \ mod 7.$$

E como a cada dia que se passa somamos 1 dia, definimos aqui $z_{d,m,Y}$ com d referente ao dia:

$$z_{d,m,Y} = d + \left[\frac{13m-1}{5}\right] + Y + \left[\frac{Y}{4}\right] - \left[\frac{Y}{100}\right] + \left[\frac{Y}{400}\right] \ \mathrm{mod}\, 7.$$

Agora, apliquemos na equação acima a mudança $Y = 100s + y$:

$$z_{d,m,Y} = d + \left[\frac{13m-1}{5}\right] + 100s + y + \left[\frac{100s+y}{4}\right] - \left[\frac{100s+y}{100}\right] + \left[\frac{100s+y}{400}\right] \ \mathrm{mod}\, 7. \quad (4.5)$$

Observemos o seguinte:

$$\begin{aligned} &\left[\frac{100s+y}{4}\right] - \left[\frac{100s+y}{100}\right] + \left[\frac{100s+y}{400}\right] \\ &= \left[25s + \frac{y}{4}\right] - \left[s + \frac{y}{100}\right] + \left[\frac{s}{4} + \frac{y}{400}\right] \\ &= 24s + \left[\frac{y}{4}\right] - \left[\frac{y}{100}\right] + \left[\frac{s}{4}\right] + \left[\frac{y}{400}\right]. \end{aligned} \quad (4.6)$$

s é a quantidade de séculos em Y, e y são os anos remascentes da divisão por 100 (portanto menor que este), então

$$\left[\frac{y}{100}\right] = \left[\frac{y}{400}\right] = 0$$

e assim, substituindo em (4.6), temos

$$24s + \left[\frac{y}{4}\right] - \left[\frac{y}{100}\right] + \left[\frac{s}{4}\right] + \left[\frac{y}{400}\right] = 24s + \left[\frac{y}{4}\right] + \left[\frac{s}{4}\right]$$

agora substituindo na equação (4.5):

$$z_{d,m,Y} = d + \left[\frac{13m-1}{5}\right] + 100s + y + 24s + \left[\frac{y}{4}\right] + \left[\frac{s}{4}\right] \ \mathrm{mod}\, 7.$$

Como $124s \equiv -2s \bmod 7$, finalizamos a chamada congruência de Zeller:

Teorema 4.1.1 (Congruência de Zeller) *Seja a data $d/m/Y$, onde d, m e Y representam dia, mês e ano, respectivamente, sendo $Y = 100s + y$, com $s \geq 16$. O dia da semana correspondente é dado por*

$$z_{d,m,Y} = d + \left[\frac{13m-1}{5}\right] - 2s + y + \left[\frac{y}{4}\right] + \left[\frac{s}{4}\right] \ mod 7.$$

Um programa em C++ aplicando a Congruência de Zeller é apresentado no **Apêndice A**.

4.2 A PÁSCOA E A FÓRMULA DE GAUSS

Primeiramente, precisamos determinar dois valores, a e b, que são os restos da divisão do ano por 4 e 7, respectivamente. Para isso, tomando o ano como $Y = 100s + y$, assim como foi feito na seção anterior. Temos então:

$$Y = 100s + y \equiv a \bmod 4 \;\Rightarrow\; y \equiv a \bmod 4$$
$$Y = 100s + y \equiv b \bmod 7 \;\Rightarrow\; 98s + y \equiv b - 2s \bmod 7 \;\Rightarrow\; y \equiv b - 2s \bmod 7.$$

Essa duas congruências lineares formam o seguinte sistema:

$$\left.\begin{array}{rcl} y & \equiv & a \bmod 4 \\ y & \equiv & b - 2s \bmod 7 \end{array}\right\} \tag{4.7}$$

Pelo Teorema Chinês dos Restos, temos $M = 4 \cdot 7 = 28$, $M_1 = 7$ e $M_2 = 4$, logo

$$y = 7 \cdot ay_1 + 4 \cdot (b - 2s)y_2 \bmod 28. \tag{4.8}$$

Temos que

$$\begin{array}{rcl} 7y_1 & \equiv & 1 \bmod 4 \;\Rightarrow\; y_1 = 3 \\ 4y_2 & \equiv & 1 \bmod 7 \;\Rightarrow\; y_2 = 2. \end{array}$$

E assim, aplicando em (4.8)

$$y \equiv 21a + 8b - 16s \bmod 28 \Rightarrow y = 21a + 8b - 16s + 28k,\ k \in \mathbb{Z}$$

que é a solução geral do sistema (4.7). Aproveitando este resultado, temos

$$\left[\frac{y}{4}\right] = 5a + 2b - 4s + 7k \Rightarrow \left[\frac{y}{4}\right] \equiv 5a + 2b - 4s \bmod 7.$$

Somando com a segunda equação de (4.7), resulta em

$$y + \left[\frac{y}{4}\right] \equiv 5a + 3b - 6s \bmod 7. \tag{4.9}$$

Vamos substituir (4.9) na fórmula de Zeller (**Teorema 4.1.1**):

$$z_{d,m,Y} = d + \left[\frac{13m - 1}{5}\right] - 2s + 5a + 3b - 6s + \left[\frac{s}{4}\right] \bmod 7$$

e como $-8s \equiv 6s \bmod 7$, então

$$z_{d,m,Y} = d + \left[\frac{13m - 1}{5}\right] + 5a + 3b + 6s + \left[\frac{s}{4}\right] \bmod 7.$$

Interessa-nos encontrar um domingo de março, logo usaremos na congruência acima $z_{d,m,Y} = 0$, $m = 1$ e $d = D$, que, quando encontrarmos, será a data da Páscoa, resultando assim em:

$$\begin{array}{rcl} \Rightarrow 0 & = & D + \left[\frac{13 \cdot 1 - 1}{5}\right] + 5a + 3b + 6s + \left[\frac{s}{4}\right] \bmod 7 \\ \Rightarrow D & = & -2 - 5a - 3b - 6s - \left[\frac{s}{4}\right] \bmod 7 \\ \Rightarrow D & = & 5 + 2a + 4b + s - \left[\frac{s}{4}\right] \bmod 7. \end{array} \tag{4.10}$$

Se a questão fosse encontrar "o sexto domingo depois de 1º março", por exemplo, já teríamos ferramentas suficientes, mas o cálculo da Páscoa não leva em conta apenas o calendário solar (para o equinócio de primavera), mas também o calendário lunar (para a lua cheia). Então temos mais o que considerar.

Meton foi um matemático e astrônomo ateniense, criador do *ciclo metônico*. Segundo esse ciclo, o tempo de 235 lunações é quase o mesmo de 19 anos trópicos. Isto é, após esse período, as fases da lua caem no mesmo dia, ou muito próximo disso. Cada ano ocupa um lugar no ciclo, e a esse lugar denominamos *número dourado*. Seja um dos ciclos começados no ano 0, então o número dourado de um ano Y é o resto c da divisão deste por 19:

$$Y \equiv c \bmod 19. \tag{4.11}$$

Outra definição a se fazer é a chamada *idade da lua*, ou, como chamaremos neste trabalho, a "*epacta*". A epacta está relacionada ao dia 1º de março. Logo num ano em que a lua nova acontece nesta data, então a epacta deste ano é 0.

Segundo a Igreja, no chamado ano 0 desta era, um dos ciclos começou e a epacta é 8. O primeiro ano lunar do ciclo tem 354 dias, e assim o ano seguinte deverá ter a epacta igual a $8 + 11 = 19$ (11 dias a mais pela diferença entre os anos solar e lunar). No ano seguinte, a epacta é $19 + 11 = 30$, nesse caso 0; subtraimos para obter um inteiro positivo menor que 30. Assim, representamos a epacta por E, onde ela seria expressa por

$$E \equiv 8 + 11c \bmod 30. \tag{4.12}$$

Ainda assim o calendário apontava as fases da lua 4 dias depois. O motivo é a disparidade entre as 235 lunações e os 19 anos solares[2] sendo necessário incluir a chamada "*equação solar*", a qual referia os anos bissextos apenas àqueles que fossem seculares múltiplos de 400. Nos outros casos (seculares não-bissextos), subtrai-se 1 das epactas. Já a denominada "equação lunar[3]" soma 1 dia a cada 300 anos, sete vezes, mais 1 dia 400 anos após isso, completando um ciclo de 2500 anos. As expressões abaixo representam as correções:

$$\begin{aligned} \text{equação solar} &\Rightarrow \left[\frac{s}{4}\right] - s; \\ \text{equação lunar} &\Rightarrow \left[\frac{8s+13}{25}\right]. \end{aligned}$$

A equação (4.12), com as expressões acima, fica assim:

$$E \equiv 8 + 11c + \left[\frac{s}{4}\right] - s + \left[\frac{8s+13}{25}\right] \bmod 30. \tag{4.13}$$

[2] 235 lunações = 6939,688 dias e 19 anos solares = 6939,602 dias, em média.

[3] Assim como foi feito na **Figura 5**, na seção anterior, é possível mostrar graficamente a validade da expressão abaixo.

A lua nova *pascal* sempre acontece no dia $31 - E$ de março. Chamaremos de N essa data:

$$\begin{aligned} N = 31 - E &\equiv 31 - 8 - 11c - \left[\frac{s}{4}\right] + s - \left[\frac{8s+13}{25}\right] \mod 30 \\ \Rightarrow N &\equiv 23 + 19c - \left[\frac{s}{4}\right] + s - \left[\frac{8s+13}{25}\right] \mod 30. \end{aligned}$$

Sabemos que a lua cheia acontece 13 dias após isso. Chamaremos essa data de P, determinada por:

$$\begin{aligned} P = N + 13 &\equiv 13 + 23 + 19c - \left[\frac{s}{4}\right] + s - \left[\frac{8s+13}{25}\right] \mod 30 \\ \Rightarrow P &\equiv 6 + 19c - \left[\frac{s}{4}\right] + s - \left[\frac{8s+13}{25}\right] \mod 30. \end{aligned}$$

A data da primeira lua cheia no dia 21 de março ou após esta data é dada por:

$$\begin{aligned} d = P - 21 &\equiv 6 - 21 + 19c - \left[\frac{s}{4}\right] + s - \left[\frac{8s+13}{25}\right] \mod 30 \\ \Rightarrow d &\equiv 15 + 19c - \left[\frac{s}{4}\right] + s - \left[\frac{8s+13}{25}\right] \mod 30 \\ \Rightarrow d &\equiv 19c + W \mod 30 \end{aligned} \tag{4.14}$$

onde d representa o resto da divisão de $19c + W$ por 30 e $W = 15 + s - \left[\frac{s}{4}\right] - \left[\frac{8s+13}{25}\right]$. Agora, vamos encontrar o primeiro domingo após essa data, então devemos, a princípio, somar 1 (para o próximo dia) a P, então

$$P = d + 22. \tag{4.15}$$

Na congruência (4.10), usando (4.14) e (4.15) obtemos:

$$\begin{aligned} e &= D - d - 22 \\ \Rightarrow e &\equiv 5 - 22 + 2a + 4b - d + s - \left[\frac{s}{4}\right] \mod 7 \\ \Rightarrow e &\equiv 2a + 4b + 6d + U \mod 7 \end{aligned} \tag{4.16}$$

onde, a partir deste ponto, tomaremos o resultado da divisão de $2a + 4b + 6d + U$ por 30 como e, sendo $U = 4 + s - \left[\frac{s}{4}\right]$. Isolando D na primeira congruência, temos a expressão final que usaremos para determinar o domingo da Páscoa:

$$D = 22 + d + e. \tag{4.17}$$

Calculamos para a Páscoa em março, mas tomamos abril como uma extensão, com as seguintes considerações:

i) se $22 + d + e \leq 31$, a Páscoa acontece em março; e

ii) se $22 + d + e > 31$, tomamos $D = d + e - 9$, e a Páscoa será em abril.

Entretanto, temos duas observações aqui, devido ao fato de que nas epactas 24 e 25, o intervalo entre duas luas novas é 29 dias, enquanto que nas outras contamos 30 dias.

A primeira é que se for epacta 24, temos:

$$\begin{aligned}
(4.13) \Rightarrow 24 &\equiv 8 + 11c + \left[\frac{s}{4}\right] - s + \left[\frac{8s+13}{25}\right] \bmod 30 \\
\Rightarrow 24 - 8 &\equiv 11c + \left[\frac{s}{4}\right] - s + \left[\frac{8s+13}{25}\right] \bmod 30 \\
\Rightarrow -16 &\equiv 19c - \left[\frac{s}{4}\right] + s - \left[\frac{8s+13}{25}\right] \bmod 30 \\
(4.14) \Rightarrow d &\equiv 15 - 16 \equiv -1 \equiv 29 \bmod 30
\end{aligned}$$

logo, $d = 29$. Neste caso, d é subtraído de 1, o que faz e somar 1. Se o resultado de $e + 1$ for 6, a data da Páscoa que seria $29 + 6 - 9 = 26$ será uma semana antes, em 19 de abril. Se $e + 1 < 6$, nada altera.

A segunda exceção é na epacta 25, o que por cálculos análogos aos de cima, acarretaria $d = 28$, e isso se o número dourado $c \geq 11$. Nada é alterado se $e \neq 6$, mas se $e = 6$, então a data da Páscoa também é antecipada em uma semana, ficando em 18 de abril.

Agora, tomando (4.7), (4.11), (4.14), (4.16), (4.17) e as duas exceções supracitadas, resumamos tudo na Fórmula de Gauss para o cômputo da Páscoa:

Teorema 4.2.1 (Fórmula de Gauss) *Seja $Y = 100s + y$ um ano maior que ou igual a 1600. A data D da Páscoa deste ano é dada pela expressão $D = 22 + d + e$ de março ou $D = d + e - 9$ de abril, onde:*

i) $Y \equiv a \bmod 4$;

ii) $Y \equiv b \bmod 7$;

iii) $Y \equiv c \bmod 19$;

iv) $19c + W \equiv d \bmod 30$;

v) $2a + 4b + 6d + U \equiv e \bmod 7$

com $W = 15 + s - \left[\frac{s}{4}\right] - \left[\frac{8s+13}{25}\right]$ *e* $U = 4 + s - \left[\frac{s}{4}\right]$. *Porém:*

1. *se $d = 29$ e $e = 6$, então a Páscoa será em 19 de abril;*

2. *se $d = 28$, $e = 6$ e $c \geq 11$, então a Páscoa será em 18 de abril.*

As expressões de W e U ficam destacadas, porque seus valores mantém-se em nos anos de um mesmo século, podendo até repetir para próximo. Por exemplo, entre os anos 1900 e 2099, temos $W = 24$ e $U = 19$. Por fim, relembrando da importância da Páscoa para a liturgia cristã, por exemplo, ela é referência para a determinação das outras festividades móveis. As mais populares, resumidas na tabela:

Festividade	**data (relativa a Páscoa), em dias**
Cinzas	46, antes
Sexta da Paixão	2, antes
Corpo de Cristo	60, depois

Tabela 3 – Principais festividades católicas relativas à Páscoa.
Fonte: [12].

além da *Quaresma*, referente ao período de 40 dias, que se inicia na quarta-feira de *Cinzas*.

4.3 A FÓRMULA DE GAUSS EM C++

Agora vamos traduzir a fórmula de Gauss, deduzida na seção anterior, para a linguagem de programação C++. Primeiramente, o programa ficaria o seguinte:

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5    int a,b,c,d,e,ano,s;
6    cout<<"\n\tALGORITMO DE GAUSS PARA O COMPUTO DA PASCOA\n\n";
7    cout<<"Insira um ano maior que ou igual a 1600: ";
8    cin>>ano;
9
10   s=ano/100;
11   a=ano%4;
12   b=ano%7;
13   c=ano%19;
14   d=(19*c+15+s-(s/4)-((8*s+13)/25))%30;
15   e=(2*a+4*b+6*d+4+s-(s/4))%7;
16
17   if(d==29 && e==6) //Excecao 1
18     cout<<"\nA Pascoa desse ano eh no dia 19 de abril.\n";
19   else
20     if(d==28 && e==6 && c>=11) //Excecao 2
21       cout<<"\nA Pascoa desse ano eh no dia 18 de abril.\n";
22     else
23       if((d+e+22)<=31)
24         cout<<"\nA Pascoa desse ano eh no dia "<<d+e+22<<" de marco.\n";
25       else
26         cout<<"\nA Pascoa desse ano eh no dia "<<d+e-9<<" de abril.\n";
```

```
27
28      return 0;
29  }
```

No início do programa, é pedido um ano maior que ou igual a 1600[4] e em seguida armazenado o seu valor na variável `ano`.

Na segunda parte do programa, são efetuados os cálculos (de acordo com Fórmula de Gauss) e registrados os valores em `s`, `a`, `b`, `c`, `d` e `e`, estes dois últimos já com as expressões de W e U inseridas.

Na próxima parte, são testadas as exceções, em três **if** *aninhados*, e com uso do comando **else**, pois garante que só efetuará uma das quatro instruções, para o ano inserido. Primeiro testa a exceção 1 (se `d`= 29 e `e`= 6), com uso do operador lógico **E** (`&&`), e, se não, testa a 2, (se `d`= 28, `e`= 6 e `c`$\geq$ 11), onde as datas serão 19 ou 18 de abril, respectivamente. Por último, caso não atenda à uma das exceções, ou seja, o caso geral, o programa testa no terceiro **if** se a data será em março (`d+e+22`$\leq$ 31) ou, se não, em abril.

Neste programa não é preciso declarar uma outra variável (que seria `D`, por exemplo, em referência a Fórmula de Gauss), pois ela só serviria para armazenar o resultado da data a ser mostrada na frase final "`A Pascoa desse ano ...`". Ao invés disso, a expressão que daria o valor de `D` é calculada e exibida junto do texto apropriado. Poderia-se ter usado essa variável, e ter sido feito o teste com relação ao mês, mas aumentaria o tamanho do programa. Contudo, da forma como está, não foi necessário reservar mais outra parte da memória para outra variável do programa. Seguem imagens do executável, calculando a data da Páscoa para o ano de 2017 (atual), 2049 e 2076 (exceções), respectivamente.

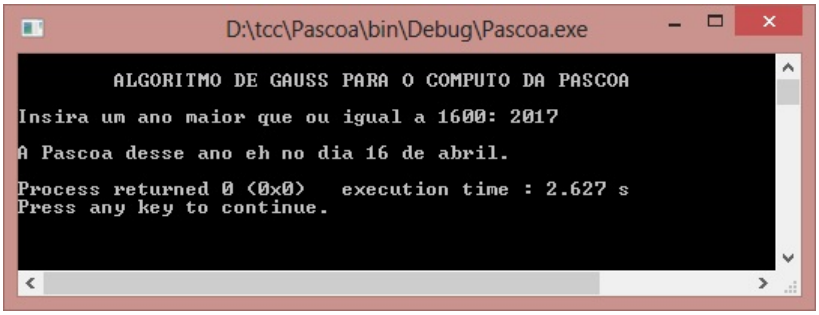

Figura 6 – Executável do programa **Pascoa**, com a data para 2017
Fonte: própria.

Este texto encerra-se com uma versão mais completa deste programa, elencando as festividades mostradas na **Tabela 3**, com exceção da *Sexta-feira da Paixão*. Por ser apenas 2 dias antes do domingo de Páscoa, o seu cálculo é trivial e dispensável, e aumentaria ainda mais o tamanho do programa.

```
1   #include <iostream>
```

[4] Devido o uso da Congruência de Zeller para a dedução da Fórmula de Gauss.

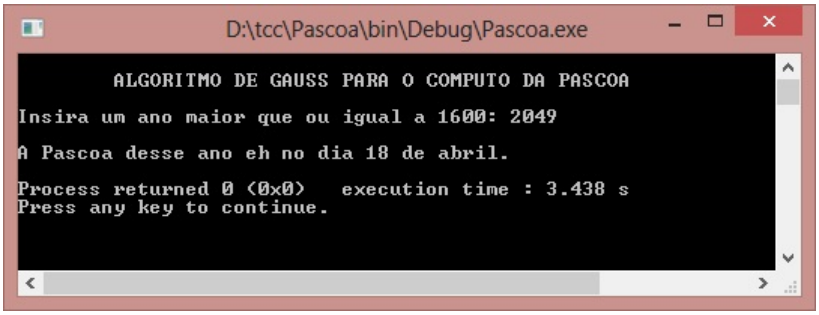

Figura 7 – Executável do programa **Pascoa**, com a data para 2049
Fonte: própria.

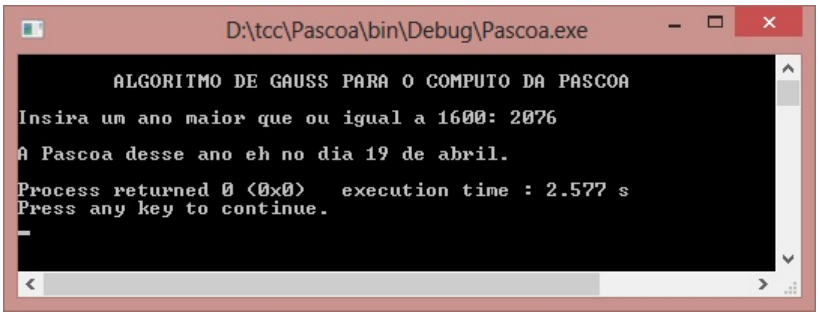

Figura 8 – Executável do programa **Pascoa**, com a data para 2076
Fonte: própria.

```
2  using namespace std;
3  int main()
4  {
5  int a,b,c,d,e,D=19,ano,s,mpas=4,mcinz=2,mcc=6,bi=0,exc=0,cinz,cc;
6    cout<<"\n\tFESTIVIDADES CATOLICAS MOVEIS\n\n";
7    cout<<"Insira um ano maior que ou igual a 1600: ";
8    cin>>ano;
9
10   s=ano/100;                              //coeficientes da
11   a=ano%4;                                //formula de Gauss
12   b=ano%7;
13   c=ano%19;
14   d=(19*c+15+s-(s/4)-((8*s+13)/25))%30;
15   e=(2*a+4*b+6*d+4+s-(s/4))%7;
16
17   if(d==29 && e==6) //Excecao 1
18     exc++;
19   else if(d==28 && e==6 && c>=11){ //Excecao 2
20     D--;
21     exc++;}
22   else if((d+e+22)<=31)  //casos gerais
23     D=d+e+22;
24   else
25     D=d+e-9;
26   if((ano%4==0 && ano%100!=0) || (ano%400==0)) //teste do dia
```

```
27        bi=1;                                          //29 de fevereiro
28    cc=D-1;         //soh muda no 3o caso
29    if(exc==1){     //teste do caso
30      cinz=D-15;
31      mcinz++;}
32    else if((d+e+22)<=31){
33      mpas--;
34      cinz=D-18+bi;
35      mcc--;}
36    else if((d+e-9)==1){
37      cinz=14;
38      cc=31;
39      mcc--;}
40    else if((d+e-9)<=15)
41      cinz=D+13+bi;
42    else{
43      cinz=D-15;
44      mcinz++;}
45
46    cout<<"\nQuarta-feira de Cinzas: "<<cinz<<"/"<<mcinz;
47    cout<<"\nPascoa: "<<D<<"/"<<mpas;
48    cout<<"\nCorpo de Cristo: "<<cc<<"/"<<mcc<<"\n";
49
50     return 0;
51  }
```

5 Considerações Finais

É esperado que um aluno de ensino médio, após a leitura deste trabalho, esteja melhor preparado nos seus estudos sobre algoritmos e linguagens de programação, e nos tópicos estudados e dúvidas recorrentes de aritmética. A verdade é que muitos alunos de graduação da área de informática, que estudam programação, tem muitas dificuldades em escrever programas simples pela insegurança em tópicos de matemática, principalmente aritmética e álgebra, que tanto se discute na escola. Este texto vem auxiliar nesse sentido, trazendo o calendário das festas móveis como um tema bem instigante, que gera perguntas e (o melhor) conjecturas a respeito do seus mecanismos aparentemente "obscuros". Os exemplos aqui tratados vem mostrar aplicações simples de matemática, e sua análise pode responder dúvidas e até remover bloqueios pré-existentes. As referências usadas para a construção desse texto tem tópicos interessantíssimos sobre História, Física, Astronomia, Teologia, Programação e é claro, a bela Matemática, portanto, são ótimas indicações.

Apêndice – Cálculo do dia da semana de uma data qualquer com C++

Vamos escrever um programa que retorne o dia da semana de uma data informada pelo usuário, que alerte sobre "bissextos inexistentes". Será usado apenas os tópicos discutidos no texto. O programa ficaria assim:

```
#include <iostream>
using namespace std;
int main()
{
   int dia,mes,ano,z,bi=0,y,c;
   cout<<"Digite uma data valida, lembrando dos anos bissextos!\n";
   cout<<"Comecando pelo dia: ";
   cin>>dia;
   cout<<"\nAgora o mes: ";
   cin>>mes;
   cout<<"\nPor fim o ano: ";
   cin>>ano;
   if((ano%4==0 && ano%100!=0) || (ano%400==0)) //teste do dia "29"
      bi=1;
   if(bi==0 && dia==29 && mes==2)
      cout<<"\n\nFevereiro nao tem dia 29 para o ano inserido!\n\n";
   else{
      if(mes<=2){   //"Ajuste do mes"
         mes+=10;
         ano--;
      }
      else
         mes-=2;

      c=ano/100;
      y=ano%100;

      z=(dia+((13*mes-1)/5)-2*c+y+(y/4)+(c/4))%7;

      switch(z){
         case 0:
            cout<<"\n\nDomingo\n\n";
            break;
         case 1:
            cout<<"\n\nSegunda-feira\n\n";
            break;
         case 2:
```

```
38              cout<<"\n\nTerca-feira\n\n";
39              break;
40          case 3:
41              cout<<"\n\nQuarta-feira\n\n";
42              break;
43          case 4:
44              cout<<"\n\nQuinta-feira\n\n";
45              break;
46          case 5:
47              cout<<"\n\nSexta-feira\n\n";
48              break;
49          case 6:
50              cout<<"\n\nSabado\n\n";
51              break;
52      }
53   }
54   return 0;
55 }
```

O programa usa explicitamente a fórmula de Zeller com as variáveis `dia`, `mês` e `ano`, armazenando o seu valor na variável `z`.

Na linha 29 começa o "teste do dia 29". Na primeira parte, `ano` deve ser múltiplo de 4 **E NÃO** deve ser de 100, **OU** então deve ser múltiplo de 400, condições essas para o ano ser bissexto, e sendo satisfeitas, atribuem a `bi` o valor 1, para testar o próximo **if**. Caso não seja bissexto, `bi` continua com o valor 0 (pré-inicializado), e o programa testa a próxima condição, fundamental para o resto do programa. Caso as três condições sejam verdadeiras (`b` = 0, `dia` = 29 e `mes` = 2), o programa acusa que o ano declarado não é bissexto, e portanto, não aceita dia 29 de fevereiro, e encerra.

Observa-se que o restante do programa só acontecerá se o teste anterior resultar em *falso*, pois não faria sentido não ter o dia 29 de fevereiro do ano em questão, e mesmo assim o programa retornar um dia da semana[1]. Isso se deve ao uso do comando **else**, como explicado na **Seção 3.3**. Se passar neste teste, o programa deve "ajustar" o mês para o formato que a fórmula de Zeller exige (março = 1, abril = 2,..., com janeiro e fevereiro pertencendo ao ano anterior). Assim, se for `mes` igual a 1 ou 2, soma-se 10, e `ano` subtrai 1 de seu valor. Se não for (`mes` $\geq$ 3), apenas reduz-se em 2.

Em seguida, o programa armazena em `c` a parte inteira da divisão de `ano` por 100 (número de séculos), e em `y` o resto dessa divisão (anos remanescentes).

Então calcula o dia pela fórmula de Zeller, armazenando-o em `z`. Depois usa o comando **switch**, que analisa cada uma das possíveis respostas. Isso também poderia

[1] Por definição, a fórmula de Zeller retorna um valor entre 0 e 6, inclusive, para valores válidos de dia, mês e ano, mas, na prática, retornará um dos valores para qualquer inteiro inserido nas variáveis. Para o C++, em especial para o programa acima, retornaria tais valores até para variáveis decimais! A nossa garantia, pelo menos para este último caso, é a definição das variáveis como sendo **inteiras**; um decimal lido, será armazenado apenas a sua parte inteira.

ser feito com vários comandos **if** aninhados. O **case default** não foi necessário neste programa, pois `z` só possui os valores de 0 a 6 como possíveis resultados. Abaixo, uma imagem do executável, testando a data 30/07/2017.

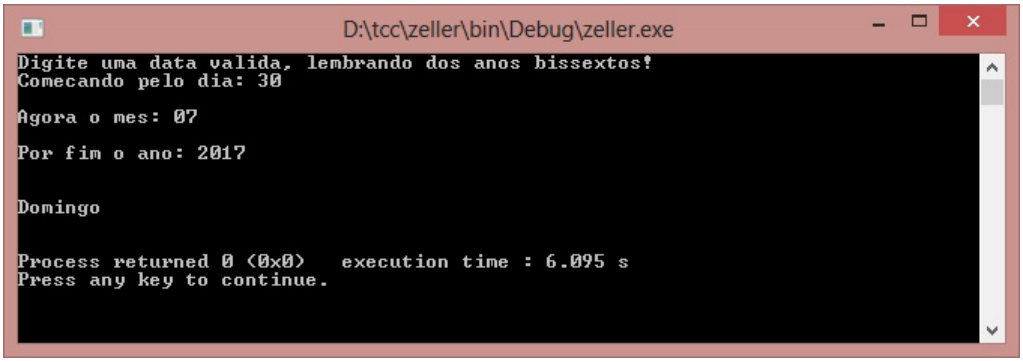

Figura 9 – Executável do programa **zeller**
Fonte: própria.

REFERÊNCIAS

[1] ALENCAR FILHO, Edgard de. **Teoria elementar dos números**. São Paulo: Nobel, 1981.

[2] ALEXANDER, Amir. **Infinitesimal:** A teoria matemática que revolucionou o mundo. Rio de Janeiro: Zahar, 2016.

[3] ALGORITMOS E ESTRUTURAS DE DADOS I. Disponível em: <http://homepages.dcc.ufmg.br/ rodolfo/aedsi-2-10/ComputadorSimplificado/computador-simplificado.html>. Acesso em: 21 maio 2017.

[4] CHERMAN, Alexandre. **O tempo que o tempo tem:** Por que o ano tem 12 meses e outras curiosidades sobre o calendário. Rio de Janeiro: Zahar, 2008.

[5] DAVIS, Stephen R. **Começando a programar em C++ para leigos**. Rio de Janeiro: Alta Books, 2011. (Coleção Para Leigos)

[6] DE ABREU, Maria Youssef; DURÃO, Adja Balbino DE A. B. **Um Pequeno Glossário Terminológico Bilíngue Hebraico-Português da Páscoa Judaica**. *Signum: Estudos da Linguagem*. vol. 9. n. 2. p. 27–43. 2006.

[7] HEFEZ, Abramo. **Aritmética**. 1. ed. 2. reimp. Rio de Janeiro: SBM, 2014. (Coleção PROFMAT)

[8] MACHADO, Rubens. **Data da páscoa e ano bissexto**. Disponível em: <http://www.astro.iag.usp.br/ rgmachado/other/pascoa.html>. Acesso em: 05 maio 2017.

[9] MARQUES, Manuel Nunes. **Origem e evolução do nosso calendário**. Disponível em:<http://www.mat.uc.pt/ helios/Mestre/H01orige.htm>. Acesso em: 30 maio 2017.

[10] MIZRAHI, Victorine Viviane. **Treinamento em linguagem C++, módulo 1**. 2. ed. São Paulo: Pearson Prentice Hall, 2006.

[11] PAPA GREGÓRIO XIII. *INTER GRAVISSIMAS*. Disponível em: <http://www.bluewaterarts.com/calendar/NewInterGravissimas.htm>. Acesso em: 05 maio 2017.

[12] PARÓQUIA SANTO ANTÔNIO DE PÁDUA. Disponível em: <http://www.psantoantonio.org.br/calendario.htm>. Acesso em: 01 junho 2017.

[13] PAULINAS. **Páscoa**. Disponível em: <http://www.ecclesia.pt/catolicopedia/>. Acesso em: 14 maio 2017.

[14] PEREIRA, José Ilhano da Silva. **Teoria dos Números e a Páscoa**. In: XIII SEMANA DE INICIAÇÃO CIENTÍFICA DA URCA. *Anais...* 2011. Crato: URCA, 2011.

[15] SIDKI, Said. **Introdução à Teoria dos Números**. 1. ed. Rio de Janeiro: IMPA, 1975.