INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

EXPLORANDO O PENSAMENTO COMPUTACIONAL:
proposta de atividades envolvendo a divisão euclidiana

Thiago Barroso Fonte Boa

Dissertação apresentada ao Programa de Pós-Graduação *Stricto Sensu* Mestrado Profissional em Matemática em Rede Nacional (PROFMAT), como parte dos requisitos para obtenção do título de Mestre em Matemática, orientada pela Profa. Dra Flávia Milo dos Santos.

IFSP
São Paulo
2022

Catalogação na fonte
Biblioteca Francisco Montojos - IFSP Campus São Paulo
Dados fornecidos pelo(a) autor(a)

b662e Boa, Thiago Barroso Fonte
Explorando o pensamento computacional: proposta de atividades envolvendo a divisão euclidiana / Thiago Barroso Fonte Boa. São Paulo: [s.n.], 2022.
104 f. il.

Orientadora: Dra. Flávia Milo dos Santos

Dissertação (Mestrado Profissional em Matemática em Rede Nacional) - Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, IFSP, 2022.

1. Pensamento Computacional. 2. Resolução de Problemas. 3. Algoritmos. 4. Scratch. 5. Linguagem C. I. Instituto Federal de Educação, Ciência e Tecnologia de São Paulo II. Título.

CDD 510

THIAGO BARROSO FONTE BOA

EXPLORANDO O PENSAMENTO COMPUTACIONAL:
proposta de atividades envolvendo a divisão euclidiana

Dissertação apresentada e aprovada em 15 de dezembro de 2022 como requisito parcial para obtenção do título de Mestre em Matemática.

A banca examinadora foi composta pelos seguintes membros:

Profa. Dra Flávia Milo dos Santos
IFSP – Câmpus São Paulo
Orientadora e Presidente da Banca

Prof. Dr. Marco Aurélio Granero Santos
IFSP – Câmpus São Paulo
Membro da Banca

Prof. Dr. Marcio Vieira de Almeida
IFSP – Câmpus São Paulo
Membro da Banca

Profa. Dra Celina Aparecida Almeida Pereira Abar
PUC – Câmpus São Paulo
Membro da Banca

Prof. Dr. Rogério Galante Negri
Unesp – Câmpus São José dos Campos
Membro da Banca

“Todos temos duas vidas. A segunda começa quando percebemos que só temos uma.”

Confúcio

Para Fer, João e Laura. Com vocês, tudo é possível.

AGRADECIMENTOS

Agradeço aos meus pais, Francisca e Antonio, que sempre acreditam em tudo que me proponho a fazer, mesmo quando não sou capaz.

Aos meus colegas do mestrado pelas conversas e incentivo que ocorriam principalmente durante nossos horários de café.

Aos meus professores da Universidade Estadual de Campinas que me incentivaram a fazer o mestrado e continuar buscando mais conhecimento.

Aos professores do Instituto Federal de São Paulo, pela dedicação, empenho e preocupação que reiteradamente tiveram em nos auxiliar, e pelos caminhos que nos iluminaram. Essa equipe é de altíssimo nível.

À minha orientadora, Flávia Milo dos Santos, por todo o tempo e paciência dedicados, por toda a motivação e envolvimento, e também pelo carinho comigo e com minha família. Flávia, não há palavras suficientes para agradecer.

À minha “marida”, Fernanda, pelo companheirismo, pela crença em mim, pelo apoio incondicional, pelo suporte, pelo João e pela Laura, enfim por tudo. Fer, sem você minha vida não teria nenhum sentido.

RESUMO

Este trabalho tem por objetivo explorar o pensamento computacional na resolução de problemas. Baseado em quatro pilares identificados por decomposição, abstração, reconhecimento de padrões e algoritmos, o pensamento computacional se caracteriza como importante habilidade a ser ensinada a todos. Nesse sentido, é proposta uma série de atividades explorando o pensamento computacional em problemas envolvendo o resto da divisão de números inteiros. Além disso, as atividades são propostas para serem desenvolvidas em *Scratch* e em linguagem C, explorando o potencial do desenvolvimento de algoritmos associados a uma dessas linguagens de programação. O pensamento computacional se configura como processo mental associado à formulação e resolução de problemas, sendo de grande importância na construção do conhecimento, permitindo uma análise do problema e da solução e possibilitando a busca por soluções mais eficientes e efetivas. Busca-se, portanto, evidenciar neste trabalho a importância do pensamento computacional, ressaltando-se seus aspectos principais.

Palavras-chaves: Pensamento computacional, Resolução de problemas, Algoritmos, Scratch, Linguagem C.

ABSTRACT

This study aims to explore computational thinking in problem-solving. Based on four principles decomposition, abstraction, pattern recognition and algorithms, computational thinking is characterized as an important skill to be taught to everyone. In this sense, a set of activities are proposed exploring computational thinking in problems involving the remainder of the division of integers. In addition, the activities are proposed to be developed in Scratch and in C language, exploring the potential of algorithm development associated with one of these programming languages. Computational thinking is characterized as a mental process associated with the formulation and the resolution of problems, being of great importance in the construction of knowledge, allowing an analysis of the problem and the solution and enabling the search for more efficient and effective solutions. The aim is, therefore, to highlight in this study the importance of computational thinking, emphasizing its main aspects.

Keywords: Computational thinking, Problem-solving, Algorithms, Scratch, C language

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. Introdução

O ensino é repleto de desafios. Alguns desses desafios surgem da falta de interesse demonstrada pelos alunos, outros das dificuldades no entendimento dos assuntos abordados ou advindos da metodologia utilizada para abordar alguns conceitos.

No entanto, esses desafios são o que tornam o ensino tão interessante e instigante. A possibilidade de despertar um olhar diferenciado do aluno em relação àquele conteúdo ou a perspectiva de uma abordagem diferenciada são grandes motivações na prática docente. Nesse sentido, faz-se sempre necessária a busca por maneiras diferentes de explorar um conceito ou uma habilidade.

Em relação ao ensino de matemática e tomando por referência a Base Nacional Comum Curricular (BNCC), (BRASIL (2018)), há a proposta de que os estudantes utilizem tecnologias desde os primeiros anos do Ensino Fundamental. Essa abordagem possibilitaria o desenvolvimento de um pensamento computacional quando esses estudantes atingissem os anos finais. Esse pensamento computacional se caracterizaria como um conjunto de habilidades necessárias para a resolução de problemas. As tecnologias e o pensamento computacional propostos pela BNCC envolvem o uso de planilhas eletrônicas, bem como a elaboração de algoritmos.

Dessa forma, a BNCC destaca a importância das tecnologias digitais para a investigação matemática durante o Ensino Médio, além de “[...] dar continuidade ao desenvolvimento do pensamento computacional, iniciado na etapa anterior” (BRASIL, 2018, p.528).

A investigação matemática se propõe a promover um maior envolvimento criativo e reflexivo, por parte dos alunos, nas aulas de matemática. Durante uma aula de investigação matemática, o professor deve desempenhar os papéis de desafiar, apoiar, avaliar, informar e promover a reflexão (PONTE *et al.*, 1998). Assim, o professor deve instigar seus alunos em busca do saber.

Com o intuito de se valer do envolvimento que a investigação matemática pode propiciar, e da forma como os conceitos matemáticos podem ser trabalhados através da identificação e do uso do pensamento computacional, nesse trabalho serão apresentadas atividades associadas ao conceito de divisão euclidiana.

Nessas atividades buscam-se, a princípio, a identificação de padrões, a determinação da estratégia matemática de resolução do problema e a estruturação

de um esquema computacional, seja ele por um algoritmo ou mesmo por uma sequência de passos de modo menos formal.

O objetivo principal será apresentar conceitos ou problemas com abordagens que possibilitem visões alternativas para os alunos, podendo contribuir para uma participação mais efetiva desses alunos na construção do seu conhecimento, utilizando duas linguagens de programação nas atividades, *Scratch* e linguagem C, além de destacar a importância do desenvolvimento do pensamento computacional como habilidade fundamental. Como afirma Wing (2006), o pensamento computacional é uma habilidade para todos.

Este trabalho está organizado em quatro capítulos e dois apêndices.

O segundo capítulo trata de conceitos acerca do pensamento computacional, da resolução de problemas, de lógica e linguagens de programação.

O terceiro capítulo se encarrega de retomar alguns conceitos sobre divisão inteira, necessários nas atividades a serem desenvolvidas. Além disso, o terceiro capítulo apresenta uma série de atividades que se relacionam com conceitos de divisão inteira. Essas atividades encontram-se organizadas de modo a desenvolver o pensamento computacional. Nelas há a proposta de resolução utilizando o *Scratch* e a linguagem C.

O quarto capítulo traz as considerações finais, além de propostas de trabalhos futuros.

No Apêndice A é feita uma explicação geral sobre os blocos do *Scratch* e seus operadores, bem como dos blocos que foram utilizados nas propostas de resoluções das atividades, além de exemplos de estruturas condicionais e de repetição.

O Apêndice B se ocupa de trazer uma explanação sobre as instruções, funções e estruturas da linguagem C e de sintaxes que foram utilizadas nas soluções propostas nas atividades. Esse apêndice também se ocupa de mostrar possíveis estruturas condicionais e de repetição na linguagem C, e de indicar algumas possibilidades de compilação de programas com ou sem a necessidade de instalação de *software*.

2. Fundamentação teórica

Nesse capítulo são feitos levantamentos acerca do pensamento computacional, destacando seu papel na resolução de problemas e como habilidade a ser desenvolvida. Também são ressaltadas algumas características sobre lógica e linguagens de programação com o intuito de solucionar problemas. As atividades que serão propostas nesse trabalho utilizam-se de recursos computacionais. Cada atividade sugerida apresenta codificações em duas linguagens de programação: uma em *Scratch* e outra em linguagem C.

2.1. Pensamento computacional e resolução de problemas

A cultura digital tem influenciado grandes mudanças nas sociedades contemporâneas, como destacado pela BNCC. Com um maior acesso às tecnologias digitais por meio de computadores, *smartphones* e afins, os estudantes encontram-se cada vez mais atuantes nessa cultura.

Turkle e Papert (1990) propõem que o computador seria um catalisador de mudanças, não apenas na cultura computacional, mas na cultura em geral. Os autores ainda destacam a importância dos diferentes saberes e das diferentes formas de pensar que essa cultura possibilita. Levando-se em conta esse panorama, faz-se necessário que a escola também se aproprie de novas formas de linguagem e de manipulação “[...] e que eduque para usos mais democráticos das tecnologias e para uma participação mais consciente na cultura digital” (BRASIL, 2018, p.61).

A BNCC apresenta, nas competências gerais da educação básica, a necessidade da utilização de diferentes linguagens – podendo ser verbal, corporal, visual, sonora e digital – para expressão e compartilhamento de informações e experiências, ideias e sentimentos. Ainda nas competências gerais da educação básica, há uma proposta de compreensão, utilização e criação de tecnologias digitais, com o intuito de produzir conhecimento e resolver problemas, propiciando ao aluno a chance de atuar como protagonista em sua vida pessoal e coletiva.

Dentre as competências específicas de matemática para o ensino fundamental, pode-se destacar a capacidade de enfrentar situações-problema em diversos contextos, expressando respostas e sintetizando soluções, “[...] utilizando diferentes

registros e linguagens (gráficos, tabelas, esquemas, além de texto escrito na língua materna e outras linguagens para descrever algoritmos, como fluxogramas, e dados)" (BRASIL, 2018, p.267). Neste ponto, nota-se a perspectiva do desenvolvimento do pensamento computacional aliado a formas alternativas de linguagem, seja utilizando algoritmos, seja utilizando fluxogramas ou mesmo descrições, na língua materna, de passos a serem seguidos.

De acordo com Wing (2006), uma das características do pensamento computacional é que ele seja uma maneira que os seres humanos utilizam para a resolução de problemas, e não uma forma de tentar fazer com que os humanos raciocinem como máquinas. Segundo a autora, enquanto os computadores são chatos e entediantes, seres humanos são espertos e imaginativos. A autora ainda destaca que o pensamento computacional representa uma atitude de aplicação generalizada e um conjunto de habilidades que todos, sem exceção, deveriam estar ávidos por aprender e utilizar.

Em uma revisão da literatura realizada por Selby e Woollard (2013), nota-se que existe um consenso em se estabelecer que o pensamento computacional seja um processo de pensamento, incorporando um conjunto de ferramentas mentais. Essas ferramentas seriam utilizadas para transformar problemas de difícil solução em problemas mais fáceis de serem resolvidos.

Associada a essa ideia de transformação de problemas encontra-se o conceito de resolução de problemas, estabelecendo-se que o pensamento computacional seja o processo mental associado à formulação e resolução de problemas que possam ser expressos através de passos de resolução, ou de algoritmos.

Para Lu e Fletcher (2009), da mesma forma que temas como representação da informação, abstração, eficiência e heurística são recorrentes em atividades humanas cotidianas e não restritas a algumas carreiras ou a habilidades específicas de uma área, o mesmo ocorreria com o pensamento computacional. Wing (2006) destaca que o pensamento computacional é uma habilidade fundamental para todos, e não apenas a cientistas da computação.

Gal-Ezer e Harel (1998) enfatizam que a ciência da computação não é apenas a base científica da principal revolução tecnológica, e sim que em seu cerne há uma forma especial e poderosa de pensar, essencial para lidar com os problemas complexos presentes no mundo atual.

Assim, a proficiência em pensamento computacional ajudaria a, sistematicamente, processar informações e tarefas, de forma correta e eficiente (ABELSON; SUSSMAN, 1996). O termo ciência da computação teria pouco a ver com computadores. A revolução computacional estaria mais relacionada à maneira que pensamos e à forma que expressamos o que pensamos.

Nesse sentido, pode-se novamente observar que a BNCC destaca, também nas competências gerais da educação básica, a importância do exercício da curiosidade intelectual, se utilizando da investigação e da reflexão, da imaginação e da criatividade, em busca de formular e resolver problemas, obtendo soluções.

Uma das ações propostas seria “[...] selecionar, produzir, aplicar e avaliar recursos didáticos e tecnológicos para apoiar o processo de ensinar e aprender”, além de “[...] conceber e pôr em prática situações e procedimentos para motivar e engajar os alunos nas aprendizagens” (BRASIL, 2018, p.17). Assim, novas práticas se fazem necessárias na busca de um maior engajamento dos alunos na construção do conhecimento. Diferentes abordagens podem propiciar um envolvimento mais efetivo desses alunos. Com esse intuito, pode-se observar a importância de uma das características do pensamento computacional, destacada por Wing (2006), que é a necessidade do desenvolvimento de um pensamento em diferentes níveis de abstração.

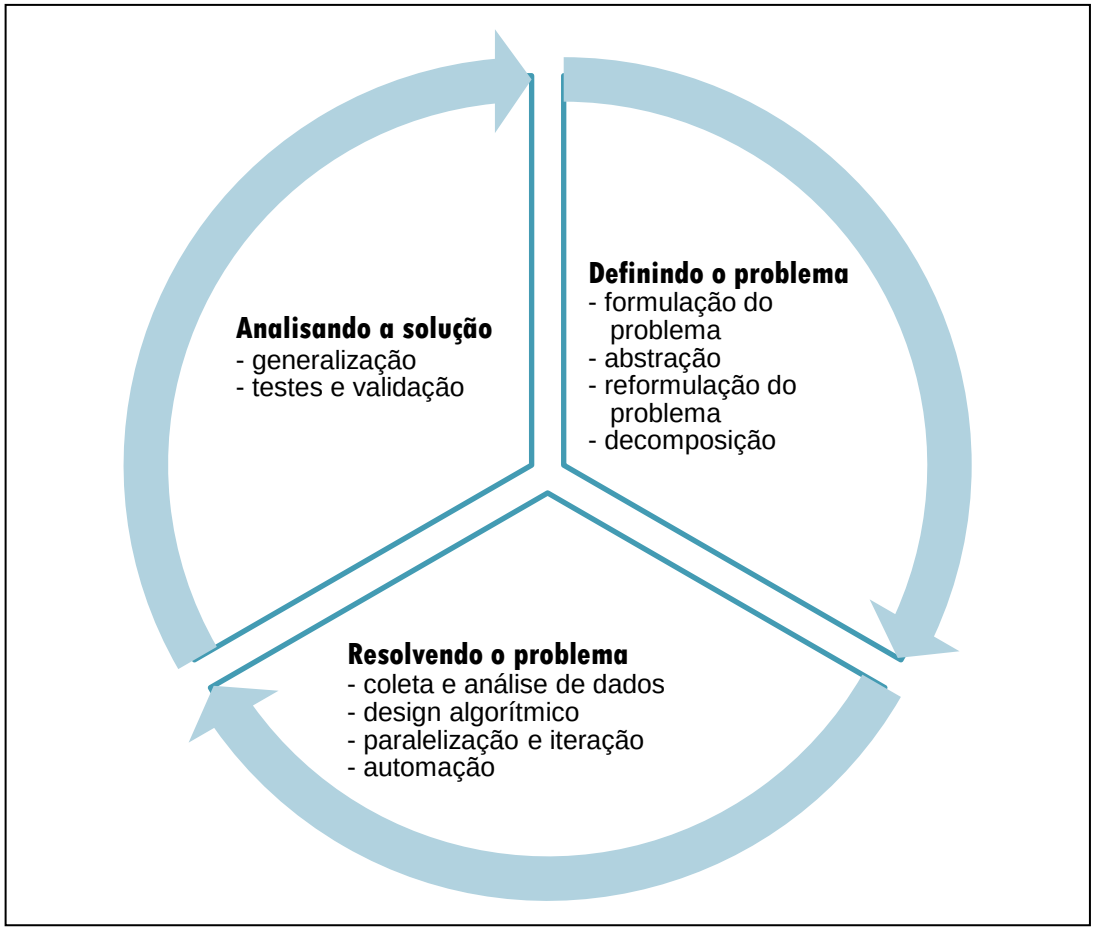
Wing (2006) ainda ressalta que o pensamento computacional deve se combinar e complementar com o pensamento matemático. Para Selby e Woollard (2013) o pensamento computacional estaria relacionado ao pensamento matemático bem como a um pensamento lógico.

A BNCC aponta ainda a necessidade do desenvolvimento do letramento matemático. Esse letramento estaria associado às “[...] competências e habilidades de raciocinar, representar, comunicar e argumentar matematicamente” (BRASIL, 2018, p.266), favorecendo a formulação e a resolução de problemas. Para Guarda e Pinto, em sua revisão bibliográfica acerca do pensamento computacional, este pode ser visto “[...] como uma maneira de resolver problemas algoritmicamente” (GUARDA; PINTO, 2020, p.1469).

Como propõem Palts e Pedaste (2020), o pensamento computacional pode ser estruturado como um modelo cíclico de resolução de problemas dividido em três estágios: definição do problema, resolução do problema e análise da solução. Palts e Pedaste (2020) também baseiam-se na ideia de que o pensamento computacional

seja uma maneira de resolver problemas de forma algorítmica. Esse modelo proposto pode ser visto na Figura 1.

Figura 1 - Modelo cíclico do pensamento computacional como resolução de problemas dividido em três estágios



Fonte: elaborada pelo autor (2022)

Nesse modelo cíclico, a solução encontrada após os três estágios pode ser aperfeiçoada caso o problema seja reformulado. Isto levaria à repetição dos três estágios até que se obtenha uma solução desejada.

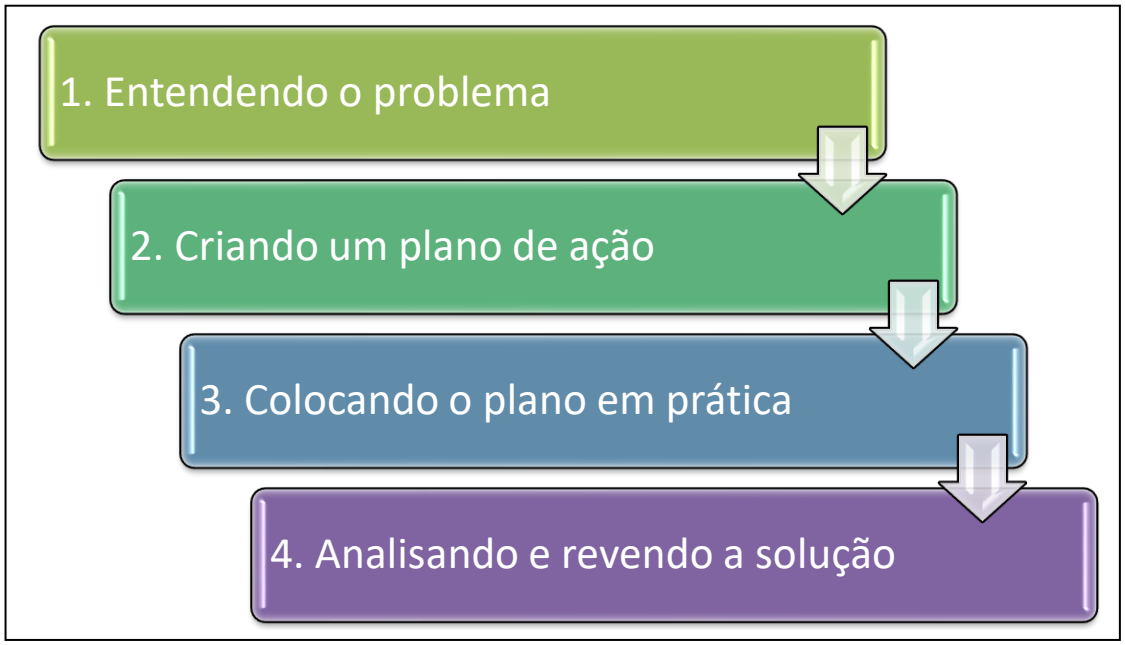
Wing (2006) enfatiza que pensar como um cientista da computação vai muito além de ser capaz de escrever um programa de computador; é essencial pensar em múltiplos níveis de abstração. O pensamento computacional, segundo Wing (2014), engloba os processos de pensamentos envolvidos na formulação de um problema e na expressão de sua solução ou soluções, de tal modo que um computador – sendo este um humano ou uma máquina – seja capaz de executá-los de forma eficiente. Logo, o pensamento computacional pode se combinar com elementos do pensamento

matemático, seja através de atividades direcionadas para o uso de tecnologias, seja em atividades associadas à resolução de problemas.

Polya (1944) é uma referência em sistematizar a forma de resolver problemas e de estruturar estratégias que levassem a resoluções de problemas. Segundo Guarda e Pinto (2020), Polya com seu livro "*How to Solve It*" pode ter sido um precursor do pensamento computacional.

Em uma de suas propostas, Polya (1944) apresenta a ideia de que o trabalho de resolução de problemas possa ser dividido em quatro partes: a primeira parte trataria do entendimento do problema, preocupando-se em identificar o que é necessário encontrar; a segunda parte seria de identificação de como o desconhecido se conecta com os dados fornecidos, possibilitando a criação de um plano de ação; a terceira parte estaria associada a colocar em prática o plano de ação desenvolvido na segunda parte; finalmente, a quarta parte iria se ocupar da análise da solução completa, levando a uma revisão e discussão dessa solução. Um resumo dessas quatro fases é apresentado na Figura 2.

Figura 2 - Quatro fases da resolução de problemas proposta por Polya



Fonte: elaborada pelo autor (2022)

A quarta fase teria uma importância crucial na consolidação dos conhecimentos adquiridos pelos estudantes e em sua habilidade de resolver problemas. Seria nessa última etapa que os estudantes poderiam se dar conta da existência de outras

maneiras de resolver o problema ou mesmo da existência de outras soluções, além de poderem descobrir novos fatos acerca do problema e de sua resolução. Portanto, a quarta parte proposta por Polya (1944) contribuiria para um aperfeiçoamento da capacidade de resolver problemas.

O pensamento computacional pode ser entendido como uma série de pensamentos associados à resolução de problemas. De acordo com Wing (2006), o pensamento computacional, assim como leitura, escrita e aritmética, deveria fazer parte de um conjunto de habilidades essenciais para todas as crianças.

De acordo com Abar, Dos Santos e De Almeida (2021), o termo pensamento computacional possibilita uma nova abordagem no campo da ciência cognitiva, podendo levar ao desenvolvimento de diferentes habilidades de abstração. Nesse processo cognitivo, o pensamento computacional sistematiza os passos da resolução de problemas.

Para Liukas (2015), o pensamento computacional se define como a capacidade de pensar em problemas de tal forma que um computador seria capaz de resolvê-los. Liukas (2015) ainda destaca que o pensamento computacional é algo feito por pessoas, não computadores. Para Barr, Harrison e Conery (2011), o pensamento computacional, como processo de resolução de problemas, deve também buscar a combinação de passos e recursos mais eficientes na implementação de soluções.

Tanto Liukas (2015) quanto Brackmann (2017) estabelecem que o pensamento computacional se apoia em quatro pilares: decomposição, reconhecimento de padrões, abstração e algoritmos.

Esses quatro pilares podem ser brevemente descritos conforme apresenta Brackmann (2017, p.33):

> O Pensamento Computacional envolve identificar um problema complexo e quebrá-lo em pedaços menores e mais fáceis de gerenciar (DECOMPOSIÇÃO). Cada um desses problemas menores pode ser analisado individualmente com maior profundidade, identificando problemas parecidos que já foram solucionados anteriormente (RECONHECIMENTO DE PADRÕES), focando apenas nos detalhes que são importantes, enquanto informações irrelevantes são ignoradas (ABSTRAÇÃO). Por último, passos ou regras simples podem ser criados para resolver cada um dos subproblemas encontrados (ALGORITMOS).

Dessa forma, o pensamento computacional irá se ocupar de abordar um problema tentando decompô-lo em problemas menores ou de mais fácil resolução (decomposição), dentro dos quais pode haver a identificação de algum método ou tática de resolução já conhecidos (reconhecimento de padrões).

O pensamento computacional ainda seria responsável por filtrar quais são as informações relevantes e quais devem ser descartadas (abstração), sendo assim capaz de definir quais passos devem ser seguidos para o estabelecimento de formas de resolver os problemas menores em que o problema original foi dividido (algoritmos) e, por conseguinte, resolvendo o problema inicial.

Na decomposição, a preocupação é em dividir o problema em partes menores. Ela pode ser vista como uma espécie de modularização. Logo, olhando para partes menores é possível ter mais atenção a detalhes, simplificando a solução de cada uma dessas partes. Segundo Shute, Sun e Asbell-Clarke (2017), é necessário assegurar que cada parte da resolução de problemas menores seja feita de forma eficiente e sem pendências.

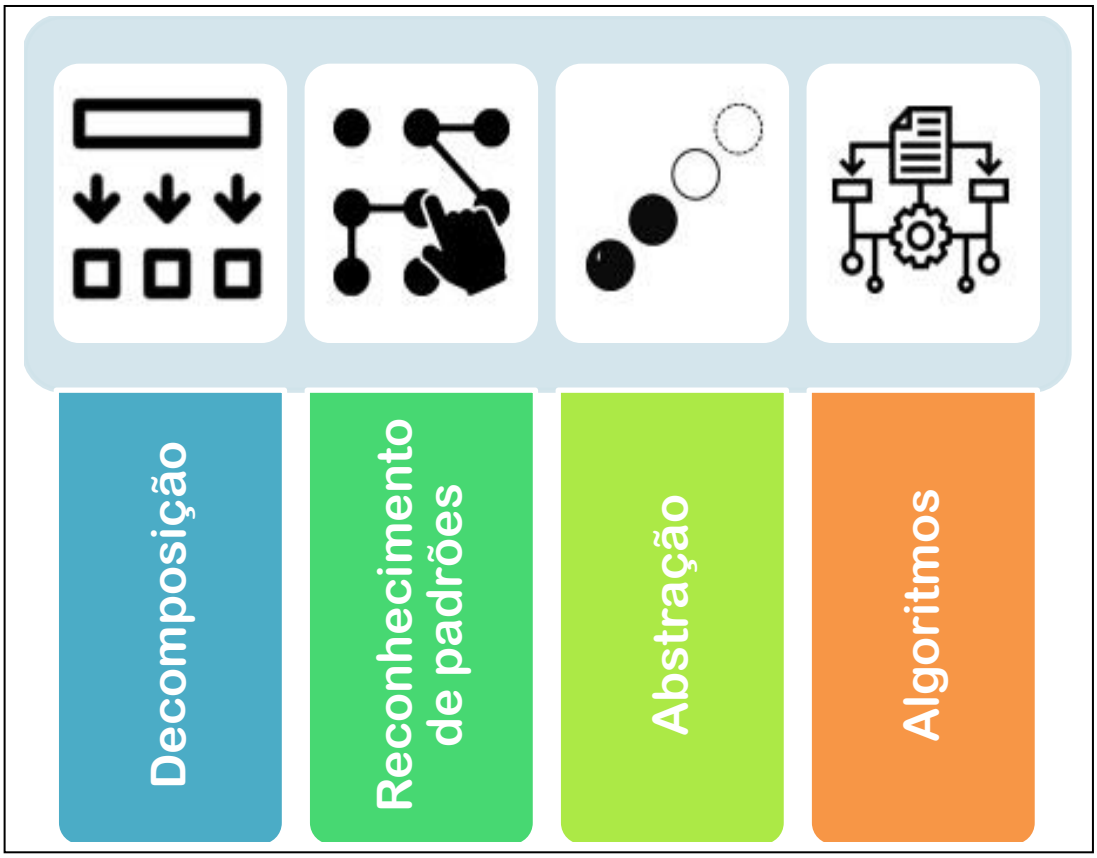
O reconhecimento de padrões engloba a identificação de similaridades com outros tipos de estratégias já conhecidas ou já utilizadas em outros problemas. Tenta-se estabelecer uma generalização do problema para se obter uma solução.

A abstração se encarrega de eliminar o que não é necessário. Aqui há a necessidade de se criar uma representação do que deve ser resolvido. Portanto deve-se eliminar detalhes que não sejam significativos para que se foque no que for essencial.

O pilar algoritmos está diretamente ligado à criação de um plano, de uma sequência de passos ou de um conjunto de instruções que sejam capazes de conduzir à resolução do problema.

Os quatro pilares em que se apoia o pensamento computacional, como discutidos anteriormente, podem ser visualizados na Figura 3.

Figura 3 - Os quatro pilares do pensamento computacional



Fonte: elaborada pelo autor (2022)

Papert (1980) descreve um processo de habilidade mental que permitiria às crianças o desenvolvimento de competências através de uma prática de programação. Dessa maneira, o pensamento computacional seria capaz de auxiliar nas formas de pensamento e de acesso ao conhecimento, conduzindo a uma melhoria nas habilidades associadas à resolução de problemas.

De acordo com Van de Walle (2001), a maioria dos conceitos matemáticos importantes, senão todos, podem ser melhor ensinados através da abordagem da resolução de problemas. Van de Walle (2001) também destaca a importância dos estudantes poderem desenvolver seus próprios algoritmos na tentativa de encontrar a solução na abordagem de resolução de problemas.

De maneira informal, Wing (2014) diz que o pensamento computacional descreve a atividade mental na formulação de um problema que admita uma solução computacional. É necessário salientar que essa solução computacional independe de utilizar-se ou não máquinas: as pessoas podem aprender pensamento computacional sem máquinas (WING, 2014).

Dessa maneira, utilizando-se máquinas ou não, o pensamento computacional possibilita que novas estratégias sejam estabelecidas nas formas de pensar e de encarar um problema. A abstração desenvolvida pelo pensamento computacional permite que diferentes áreas do conhecimento sejam impactadas por essa forma de pensar.

Como bem destaca Wing (2014), os benefícios educacionais associados à capacidade de se pensar computacionalmente realçam e fortalecem habilidades intelectuais, podendo se transferir para qualquer domínio do conhecimento.

2.2. Lógica e linguagens de programação

A lógica se ocupa do estudo do raciocínio fazendo com que pensamentos tenham sentido. Para Forbellone e Eberspächer (2005), a lógica se relaciona com a forma correta do pensamento, preocupando-se em estabelecer a validade de operações, analisando as formas e leis do pensamento. A lógica de programação, por sua vez, está relacionada à forma correta de pensar que leva à criação de programas de computadores que resolvam problemas eficientemente.

Para Setzer (2002), o pensamento necessário para programar é da mesma natureza que aquele utilizado pela lógica simbólica. Forbellone e Eberspächer (2005) destacam que a lógica de programação conduz à produção de soluções que sejam válidas logicamente e que sejam coerentes, resolvendo problemas com qualidade.

O uso de uma linguagem de programação serve para motivar os estudantes, além de melhorar sua capacidade de raciocínio lógico e na resolução de problemas, como afirmam Garcia, Correia e Shimabukuro (2008). Segundo Colling et al. (2014), o ensino de programação de computadores para crianças não busca o aprendizado de uma habilidade profissional, e sim o desenvolvimento de competências lógicas com vista a melhorar outras áreas da educação.

Para França e Tedesco (2015), há a necessidade do ensino, desde a educação básica, de conceitos da Ciência da Computação para melhora do aprendizado escolar dos estudantes. Estes autores ainda destacam que esse ensino na escola pode desenvolver a capacidade de resolver problemas, apoiando e se relacionando com outras ciências.

Segundo Garlet, Bigolin e Silveira (2018), o importante é ensinar a lógica, que é a mesma para qualquer linguagem; se o estudante souber a lógica, então será fácil

aprender a sintaxe ou a estrutura de uma linguagem específica. A lógica de programação propiciará uma maior capacidade de pensar e isso ocorrerá com maior criatividade.

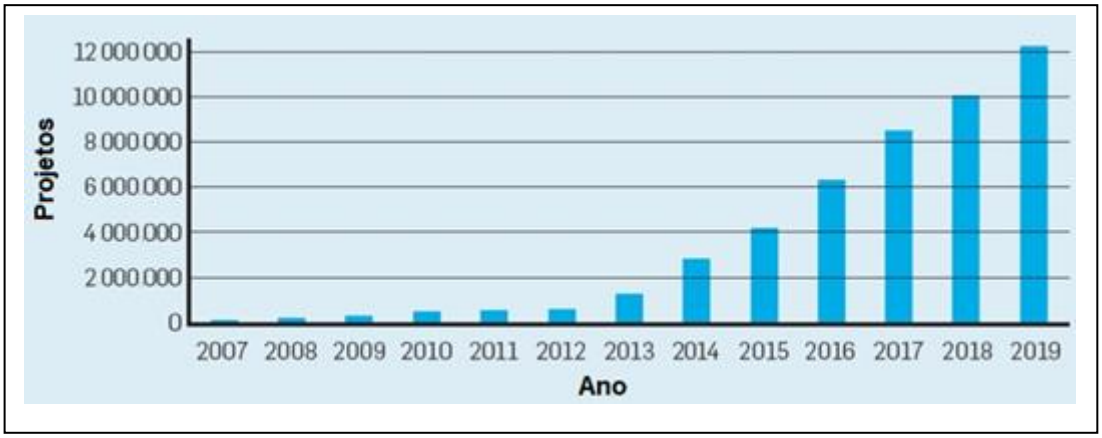
De acordo com Resnick e Rusk (2020), no processo de criar e compartilhar projetos, os estudantes não estão apenas aprendendo a escrever programas; eles estão programando para aprender. Nesse sentido, os estudantes estariam desenvolvendo uma gama de habilidades, entre elas a capacidade de resolução de problemas.

2.2.1. *Scratch*[1]

Quando foi lançado, em maio de 2007, o *Scratch* era uma aplicação que devia ser baixada e executada no computador de cada usuário. As estórias interativas, os jogos ou as animações criadas pelos usuários do *Scratch* naquele momento podiam ser compartilhadas com uma comunidade online desde que fossem carregadas no site do *Scratch*.

Em 2013, com a disponibilização de uma nova versão, o *Scratch* passou a ser executado de forma online. Isso acarretou um crescimento exponencial na comunidade *Scratch*, fazendo com que ela se tornasse a maior plataforma mundial de programação desenvolvida para crianças, disponível de forma online e gratuita. Esse crescimento pode ser observado no gráfico apresentado na Figura 4.

Figura 4 - Número de projetos em *Scratch* compartilhados por ano



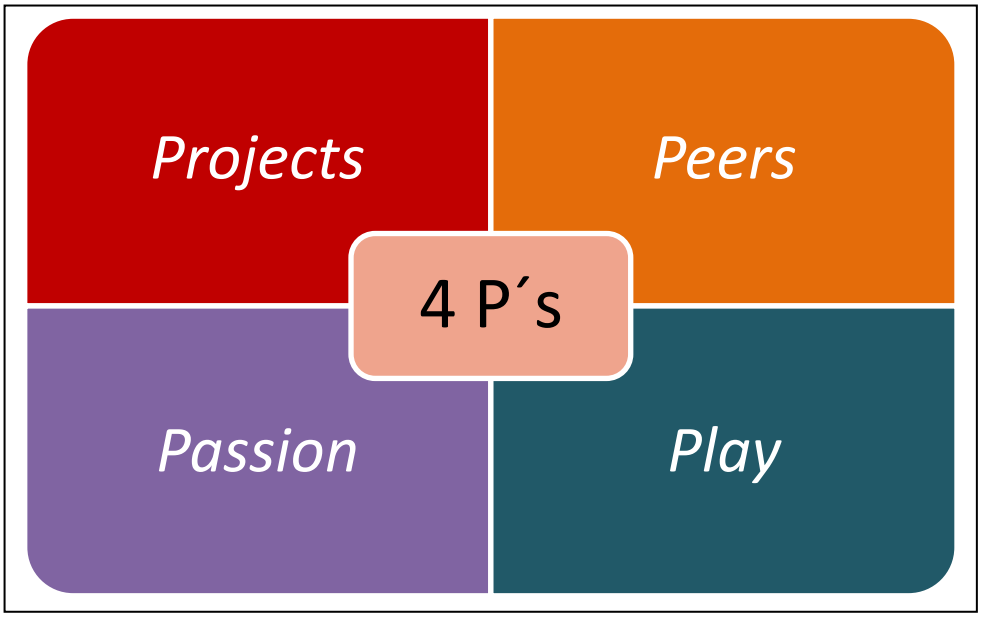
Fonte: adaptada de Resnick e Rusk (2020)

[1] Plataforma de programação baseada em blocos. Disponível em: https://scratch.mit.edu/about. Acesso em: 15 jan.2022.

Segundo Resnick e Rusk (2020), a expansão da programação aplicada à educação tem se acelerado devido ao surgimento de novas interfaces de programação; em especial, a programação baseada em blocos (*block-based coding*). Pinto (2010) destaca essa característica da programação por arrastamento de blocos de construção presente no *Scratch*, citando a criatividade como uma de suas potencialidades.

Segundo Resnick (2014), seu grupo de pesquisa no MIT *Media Lab*[2] vem trabalhando para proporcionar experiências criativas de aprendizado, de forma que as crianças possam se desenvolver como pensadores criativos. A abordagem desse grupo se baseia em quatro elementos principais, destacados como os quatro P´s do aprendizado criativo: *Projects*, *Peers*, *Passion* e *Play* (Projetos, Pares, Paixão e Preparar para brincar). Os quatro elementos principais estão destacados na Figura 5.

Figura 5 - Os quatro elementos principais do grupo de pesquisa do MIT Media Lab



Fonte: elaborada pelo autor (2022)

Dentro do elemento *Projects* destaca-se que as pessoas aprendem melhor quando trabalham ativamente em projetos que sejam significativos; o elemento *Peers* encarrega-se da importância do aprendizado de forma colaborativa e como atividade social; no elemento *Passion* ressalta-se que quando as pessoas trabalham em

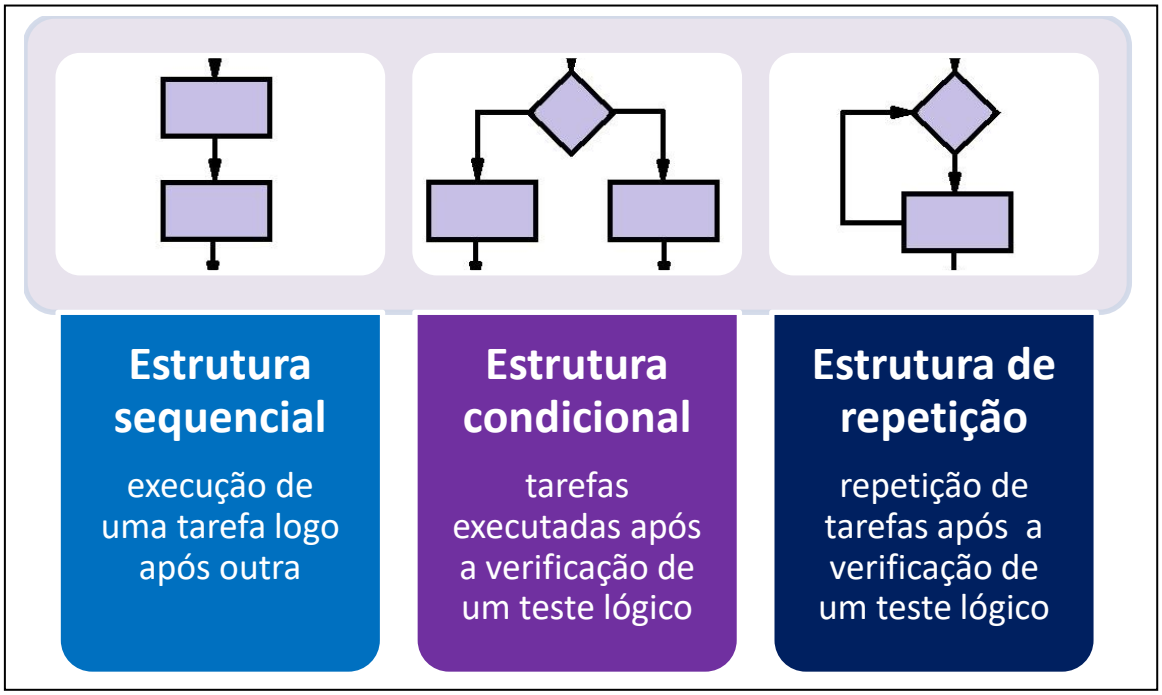
[2] Laboratório de pesquisa interdisciplinar associada a criação e desenvolvimento de tecnologia. O laboratório faz parte da faculdade de arquitetura e urbanismo do MIT (*Massachusetts Institute of Technology*). Disponível em: https://www.media.mit.edu/. Acesso em: 05 abr.2022.

projetos elas cuidam e trabalham arduamente, aprendendo mais durante o processo; por fim o elemento *Play* enfatiza que o aprendizado implica em uma experimentação lúdica. Esses quatro elementos serviram de inspiração e guia no desenvolvimento do *Scratch* como ambiente de programação (SCHMIDT; RESNICK; ITO, 2016).

2.2.2. Linguagem C

A linguagem de programação C pode ser classificada como uma linguagem estruturada. Uma linguagem estruturada estabelece que todos os programas possíveis podem ser decompostos em apenas três estruturas: estrutura sequencial, estrutura de decisão (condicional) e estrutura de repetição (laço). Essa decomposição pode ser vista na Figura 6.

Figura 6 - Três estruturas previstas em uma linguagem estruturada



Fonte: elaborada pelo autor (2022)

O uso de uma linguagem de programação estruturada é condizente com a abordagem esperada quando se pretende trabalhar com resolução de problemas, uma vez que pode-se decompor um problema em subproblemas menores. Nesse sentido, uma linguagem estruturada auxiliaria na execução do passo de decomposição que constitui um dos pilares do pensamento computacional.

Há vários livros disponíveis a respeito da linguagem C. Deitel (2006) e Schildt (2006) são boas referências para uma introdução na linguagem, entendimento de suas características e possibilidades de programação.

Programas escritos em linguagem C precisam passar por uma espécie de tradução para que possam ser executados. Esse processo é executado por um programa conhecido como compilador. Existem compiladores para a linguagem C que exigem instalação de um *software*, bem como existem compiladores *online* que não necessitam que algum *software* seja instalado. Uma breve discussão sobre algumas opções de compiladores pode ser encontrada no Apêndice B.

3. Proposta de atividades

As atividades propostas nesse trabalho tratam de discussões acerca de divisibilidade de números naturais ou inteiros. Algumas das atividades focam em problemas associados à identificação do resto da divisão euclidiana. Outras, por sua vez, necessitam da identificação do quociente, além do resto da divisão.

Existem problemas relacionados a esse tema da divisibilidade inteira que aparecem frequentemente em livros didáticos voltados ao Ensino Médio, como por exemplo em Dante (2020) ou Souza (2020), ou mesmo em materiais apostilados e exames vestibulares. Espera-se que uma abordagem desse tipo de problema com os conceitos associados ao pensamento computacional permitirá um maior envolvimento por parte dos alunos, bem como um melhor entendimento dos processos cognitivos envolvidos na busca de soluções para os problemas.

Faz-se necessário destacar que o reconhecimento de padrões, um dos pilares do pensamento computacional, é de suma importância na resolução de problemas relacionados à identificação do resto da divisão inteira. Como bem observa Wing (2006), o pensamento computacional estabelece a possibilidade de que um problema aparentemente complicado seja reformulado para algum que saibamos resolver. Nesse processo, haveria uma necessidade de transformação do problema, redução desse problema ou decomposição, outro importante pilar do pensamento computacional.

O foco das atividades aqui propostas é que sejam aplicadas a turmas de Ensino Médio. No entanto, algumas dessas atividades podem facilmente ser implementadas com alunos de Ensino Fundamental. Cabe ao professor que optar por trabalhar com alguma dessas atividades verificar a pertinência para a turma específica em que deseja aplicar tal atividade.

Soppelsa (2016) apresenta uma possibilidade de aprofundamento do conceito da divisão euclidiana para o Ensino Fundamental, oferecendo uma análise de vários livros didáticos. A sequência didática apresentada por Soppelsa (2016) se baseia na resolução de situações-problema e na identificação de relação entre o resto da divisão euclidiana e a resolução de problemas cíclicos. Essa abordagem está fortemente associada aos pilares do pensamento computacional e à resolução de problemas.

Vale ressaltar que a atividade 2, que trata de potências de i, pressupõe o conhecimento de números complexos. Portanto, entende-se que ela seria recomendada apenas a alunos de Ensino Médio que já tivessem tido contato com números complexos e potências da unidade imaginária.

Todas as atividades podem ser realizadas pensando-se no uso do pensamento computacional sem a necessidade de equipamentos eletrônicos e de internet. Essa maneira de trabalho é conhecida por forma desplugada. Assim, pode-se estabelecer que o pensamento computacional desplugado se ocupa das habilidades associadas à resolução de problemas sem o uso de máquinas. Há propostas de trabalho muito interessantes usando essa abordagem desplugada. Ressaltam-se aqui as propostas de Brackmann (2017) e Marques (2019).

A tarefa proposta por Ponte *et. al.* (1998) também poderia ser realizada utilizando-se o pensamento computacional, de forma desplugada ou não, orientado à resolução de problemas. No entanto, essa tarefa é proposta no âmbito da investigação matemática. Para Ponte, Brocardo e Oliveira (2019), a realização de uma investigação matemática envolve quatro momentos principais: (i) reconhecimento da situação, sua exploração preliminar e formulação de questões; (ii) organização dos dados e formulação de conjecturas; (iii) realização de testes e refinamento das conjecturas; e (iv) justificação das conjecturas e avaliação do raciocínio. Assim, uma investigação matemática apresenta aspectos em comum com outras atividades de resolução de problemas.

Nesse trabalho, optou-se pelo uso de linguagens de programação associadas à resolução das atividades. São exploradas duas formas de linguagens: uma baseada em construção por blocos (*Scratch*) e outra que se enquadra como linguagem estruturada (linguagem C). Outras linguagens de programação podem ser utilizadas a critério do professor.

O tempo sugerido para cada atividade é entendido como uma previsão mínima necessária para a aplicação de apenas uma das duas linguagens de programação apresentadas. Pressupõe-se que para a resolução das atividades com o uso de alguma das linguagens propostas seja necessária uma introdução acerca de lógica e da linguagem em si.

As atividades foram ordenadas da mais simples à mais elaborada, a fim de propiciar a identificação de padrões por parte dos alunos quando trabalhando com problemas relacionados a quociente e/ou resto da divisão euclidiana.

Essa ordenação também tem o intuito de mostrar uma evolução dos problemas, dos menores para os maiores, como se uma decomposição tivesse sido efetuada nessa elaboração, estimulando o processo cognitivo dos estudantes. Liukas (2015) trata de uma habilidade de quebrar grandes problemas em problemas menores, o que está em acordo com a decomposição, um dos pilares do pensamento computacional.

Em todas as atividades propostas está presente o pilar do pensamento computacional associado aos algoritmos. Todas as resoluções, sejam elas plugadas ou não, hão de se basear no estabelecimento de algoritmos, que se caracterizam como uma sequência de passos. Barr, Harrison e Conery (2011) destacam essa dimensão do pensamento computacional como sendo a automatização de soluções através de um pensamento algorítmico, caracterizado como uma série de passos ordenados.

Selby e Woollard (2013) ressaltam ainda que um algoritmo deve ser pensado como um procedimento passo-a-passo para realização de tarefas, não apenas dentro da ciência da computação, e sim em outras áreas.

As diretrizes para ensino de computação na educação básica propostas pela Sociedade Brasileira de Computação (SBC), (SBC, 2019) afirmam que o conceito de algoritmo, além de estar presente em todas as áreas, também se encontra intrinsecamente relacionado à resolução de problemas, dado que um algoritmo é a descrição de um processo.

A primeira seção desse capítulo trata de alguns conceitos sobre divisão inteira, conceitos centrais das atividades aqui propostas. As seções seguintes apresentam as seis atividades relacionadas à divisão inteira. Os apêndices presentes no final deste trabalho servem de orientação inicial aos professores sobre as linguagens abordadas. O Apêndice A apresenta uma breve discussão acerca dos blocos da linguagem *Scratch* que foram utilizados nas propostas de solução das atividades desse trabalho, com descrições dos tipos de blocos e suas características. No Apêndice B podem ser encontradas as estruturas e funções utilizadas da linguagem C, algumas sintaxes importantes para as programações, além de formas de compilar os programas com ou sem a necessidade de instalação de um *software*. Em ambos os apêndices é possível encontrar formas de uso de estruturas condicionais e de estruturas de repetição.

3.1 Alguns conceitos sobre divisão inteira

Ao se trabalhar com o conjunto dos números naturais ou com o conjunto dos números inteiros, é comum encontrar problemas relacionados ao conceito de divisibilidade. Neste trabalho, as atividades sugeridas tratam de quociente e/ou resto da divisão de números naturais. Dessa forma, faz-se necessária uma breve introdução acerca de divisibilidade, quociente e resto, além de uma visão básica sobre congruência. Os conceitos aqui apresentados foram baseados em Hefez (2016).

Um número inteiro a é dito divisível por um inteiro b ($b \neq 0$) caso exista um número inteiro c que possibilite a escrita da identidade $a = b \cdot c$. De outra maneira, diz-se que b é um divisor de a, ou ainda que b divide a, podendo ser representado pela notação $b \mid a$. Com isso, sendo a um número inteiro não nulo, é fácil verificar que:

$$i)\ 1 \mid a \qquad (a = 1 \cdot a)$$
$$ii)\ a \mid a \qquad (a = a \cdot 1)$$
$$iii)\ a \mid 0 \qquad (0 = a \cdot 0)$$

No entanto, há casos em que não existe o inteiro c que permita a escrita da identidade $a = b \cdot c$. Para esses casos, diz-se que b não divide a, representado pela notação $b \nmid a$. Quando b não divide a, pode-se utilizar o processo de divisão com resto. Hefez (2016) apresenta a demonstração do teorema que aqui será apenas enunciado.

Teorema (Divisão Euclidiana): Sejam a e b dois números inteiros com $b \neq 0$. Existem dois únicos inteiros q e r, chamados respectivamente de quociente e resto, tais que

$$a = b \cdot q + r, \text{ com } 0 \leq r < |b|.$$

Se o valor de r for zero, tem-se que $b \mid a$. Caso r seja diferente de zero, então $b \nmid a$.

Utilizando a Divisão Euclidiana é possível estabelecer o conceito de congruência. Sejam os números inteiros a, b e m, com m maior que 1. Se os restos das divisões euclidianas de a e b por m forem iguais, diz-se que a e b são congruentes módulo m. Isso pode ser representado por

$$a \equiv b\ (mod\ m).$$

Dessa definição segue de forma imediata que a congruência módulo m tem as seguintes propriedades para quaisquer inteiros a, b e c:

$i)$ Reflexiva: $a \equiv a \ (mod\ m)$;

$ii)$ Simétrica: Se $a \equiv b \ (mod\ m)$, então $b \equiv a \ (mod\ m)$;

$iii)$ Transitiva: Se $a \equiv b \ (mod\ m)$ e $b \equiv c \ (mod\ m)$, então $a \equiv c \ (mod\ m)$.

Para um inteiro positivo $m > 1$, pode-se particionar o conjunto dos inteiros em subconjuntos, cada um destes formados pelos números inteiros que têm o mesmo resto quando divididos por m. Esses conjuntos são chamados de classes de equivalência.

Assim, é possível estabelecer as seguintes classes de equivalência:

$$[0] = \{x \in \mathbb{Z} \mid x \equiv 0 \ (mod\ m)\},$$

$$[1] = \{x \in \mathbb{Z} \mid x \equiv 1 \ (mod\ m)\},$$

$$\vdots$$

$$[m-1] = \{x \in \mathbb{Z} \mid x \equiv m-1 \ (mod\ m)\}.$$

Dessa forma, se a e b têm mesmo resto r quando divididos por m, então a e b são elementos de uma mesma classe identificada por $[r]$.

A seguir são apresentados alguns exemplos.

Exemplo 1:

Tomando-se os números 12 e 3, pode-se afirmar que 12 é divisível por 3, que 3 é divisor de 12, ou ainda que 3 divide 12, representando isso por 3 | 12. Essa afirmação pode ser feita pois existe o número natural 4 que permite a escrita da identidade $12 = 3 \cdot 4$.

Exemplo 2:

Dados os números 12 e 5, é possível afirmar que 12 não é divisível por 5, que 5 não é divisor de 12, ou ainda que 5 não divide 12, isso sendo representado por $5 \nmid 12$. Chega-se a essa conclusão pois não existe número natural c tal que a equação $12 = 5 \cdot c$ seja verdadeira.

Exemplo 3:

Dividindo-se o número 17 pelo número 5, através da Divisão Euclidiana, obtém-se o quociente 3 e o resto 2, pois tem-se a identidade $17 = 5 \cdot 3 + 2$. Como o resto da

divisão de 17 por 5 foi igual a 2, pode-se dizer que o número 17 pertence à classe de equivalência $[2]$. Essa classe será composta por todos os elementos do conjunto dos inteiros que têm o resto igual a 2 quando o divisor é igual a 5.

Outros elementos da classe $[2]$ são, por exemplo, os números 12, 32 e 57. Esses três números apresentam resto 2 quando divididos por 5. Portanto são congruentes a 17 módulo 5. Pode-se representar isso por:

$$12 \equiv 17(mod\ 5),$$
$$32 \equiv 17(mod\ 5),$$
$$57 \equiv 17(mod\ 5).$$

Logo, a classe $[2]$, com relação à divisão por 5, contém todos os números que apresentam resto igual a 2 quando divididos por 5.

Há vários problemas envolvendo o reconhecimento de padrões ou a identificação de um processo cíclico que podem facilmente ser associados ao resto da divisão de números inteiros. Uma vez que cada resto na Divisão Euclidiana determina uma classe, essa classe será um conjunto cujos elementos têm mesmo resto, o que torna possível a percepção de um processo cíclico ou de um padrão. Na sequência é apresentado um exemplo de exercício que esteve presente no vestibular do ano de 2018 da Universidade Estadual do Rio de Janeiro (UERJ).[3]

Exemplo 4:

Considere a sequência $(a_n) = (2, 3, 1, -2, \dots)$, $n \in \mathbb{N}^*$, com 70 termos, cuja fórmula de recorrência é:

$$a_n = a_{n-1} - a_{n-2}.$$

Determine qual será o último termo dessa sequência.

Nesse exercício faz-se necessário estabelecer alguns termos da sequência para a identificação do padrão. Assim, tem-se:

$$a_5 = a_4 - a_3 = -2 - 1 = -3,$$
$$a_6 = a_5 - a_4 = -3 - (-2) = -1,$$
$$a_7 = a_6 - a_5 = -1 - (-3) = 2.$$

[3] Universidade do Estado do Rio de Janeiro – Vestibular Estadual 2018 – 2º Exame de Qualificação. Disponível em: https://www.vestibular.uerj.br/wp-content/uploads/2019/03/2018_2eq_prova.pdf. Acesso em: 29 jan.2022.

Logo, o sétimo termo da sequência é igual ao primeiro termo, o oitavo termo será igual ao segundo termo, e assim sucessivamente.

É possível perceber que a sequência possui 6 termos distintos que se repetem. Pode-se, portanto, estabelecer uma relação do valor de cada termo com o resto da divisão da posição ocupada pelo termo por 6.

Dessa forma, tem-se a seguinte tabela.

Tabela 1 – Relação entre o resto da divisão e o valor do termo da sequência

Resto	Termo de referência	Valor do termo
0	6º	–1
1	1º	2
2	2º	3
3	3º	1
4	4º	–2
5	5º	–3

Fonte: elaborada pelo autor (2022)

O exercício pedia que fosse encontrado o último termo da sequência. Foi dito que a sequência era constituída de 70 termos. É necessário então determinar o valor do termo da posição 70, ou seja, o valor de a_{70}.

Tomando-se a Divisão Euclidiana de 70 por 6, tem-se:

$$70 = 6 \cdot 11 + 4.$$

Como o resto da divisão de 70 por 6 é igual 4, pode-se escrever que:

$$70 \equiv 4 \ (mod\ 6).$$

Também pode-se dizer que a classe $[4]$ contém todas as posições que apresentam 4 como resto da divisão por 6, ou seja:

$$[4] \supset \{4, 10, 16, 22, 28, 34, \dots, 64, 70\}.$$

Dessa maneira, o 70º termo da sequência será igual ao 4º termo, valendo, portanto, –2.

As atividades propostas na sequência envolvem problemas associados a esses conceitos de divisibilidade inteira e ao reconhecimento de padrões em sequências utilizando estratégia relacionada à análise dos restos dos índices.

3.2. Atividade 1 – determinação do quociente e do resto em uma divisão inteira

3.2.1. Apresentação

Crie um programa para efetuar a divisão de um número inteiro por outro número inteiro. Esse programa deve pedir ao usuário a entrada de dois valores. O primeiro valor digitado pelo usuário será o dividendo e o segundo valor digitado corresponderá ao divisor. O programa deverá calcular e exibir o quociente e o resto da divisão inteira do dividendo pelo divisor.

3.2.2. Público alvo

Essa atividade pode ser trabalhada com alunos do Ensino Fundamental (séries finais) e com alunos do Ensino Médio.

3.2.3. Duração

Sugere-se um tempo de no mínimo 45 minutos para a aplicação dessa atividade.

3.2.4. Objetivo

Apresentar aos alunos duas formas de uso de programação para obtenção do quociente e do resto de uma divisão inteira. Uma das formas é através da utilização do *Scratch* e outra utilizando-se a linguagem de programação C. Será também de grande relevância, durante essa atividade, a retomada da nomenclatura associada aos elementos constituintes de uma divisão inteira, a saber, dividendo, divisor, quociente e resto.

3.2.5. Considerações ao professor e sugestão de resoluções

A atividade requer que seja realizado um algoritmo que receba como entrada dois valores: o dividendo e o divisor para o cálculo de uma divisão inteira. O algoritmo deve calcular e exibir o quociente e o resto dessa divisão. Uma resolução da atividade em *Scratch*[4] pode ser vista na Figura 7.

Figura 7 - Calculando o quociente e o resto de uma divisão inteira utilizando o *Scratch*

Fonte: elaborada pelo autor (2022)

Nessa resolução em *Scratch* foram utilizadas duas variáveis, A e B, para armazenarem, respectivamente, os valores do dividendo e do divisor fornecidos pelo usuário. Para a exibição do quociente inteiro da divisão foi utilizada a função piso (arredondamento para baixo), que retorna o maior inteiro que seja menor ou igual ao resultado da divisão de A por B.

A Figura 8 apresenta uma solução para a atividade em linguagem C[5].

[4] Disponível em: https://scratch.mit.edu/projects/693333718. Acesso em: 28 dez.2022.
[5] Disponível em: https://onlinegdb.com/IRJqoPLTu8. Acesso em: 28 dez.2022.

Figura 8 – Calculando o quociente e o resto de uma divisão inteira utilizando a linguagem C

```c
#include <stdio.h>

main(){
    int A, B;
    printf("\n\n Determinando o quociente e o resto da divisão de A por B \n\n");
    printf(" Digite o valor de A: ");
    scanf("%i",&A);
    printf(" Digite o valor de B: ");
    scanf("%i",&B);
    printf("\n\n O quociente vale %i e o resto vale %i",A/B,A%B);
}
```

Fonte: elaborada pelo autor (2022)

Essa resolução também faz uso de duas variáveis, A e B, para receberem os valores digitados pelo usuário, referentes, respectivamente, ao dividendo e ao divisor. Vale a ressalva de que na solução proposta em linguagem C não há a necessidade do uso da função piso como requerido na solução em *Scratch*, uma vez que a exibição do resultado da divisão é configurada para ser um resultado inteiro.

Tanto o *Scratch* quanto a linguagem C possuem operadores que retornam o quociente e o resto da divisão. No *Scratch* há o bloco de divisão e o bloco de resto para a determinação, respectivamente, do quociente e do resto. Na linguagem C, são utilizados os operadores / e % para cálculo, respectivamente, do quociente e do resto. Assim, essa atividade tem como propósito principal a apresentação desses operadores para que seja possível, em problemas mais complexos, uma decomposição que se utilize desses recursos em resoluções mais elaboradas. Portanto, essa tarefa já estabelece um ponto de partida para que o reconhecimento de padrões, pilar do pensamento computacional, comece a ocorrer.

3.3. Atividade 2 – potências de i

3.3.1. Apresentação

Elabore um programa que determine o resultado da potência de i (unidade imaginária no conjunto dos números complexos) dado um expoente n pertencente ao conjunto dos números inteiros. O programa deverá requerer ao usuário a entrada de um único valor: o expoente inteiro n. Dado o valor do expoente n, o programa deve determinar e exibir o resultado da potência i elevado a n.

3.3.2. Público alvo

Essa atividade pode ser trabalhada com alunos do Ensino Médio que já tenham conhecimento sobre números complexos.

3.3.3. Duração

Sugere-se um tempo de no mínimo 45 minutos para a aplicação dessa atividade.

3.3.4. Objetivo

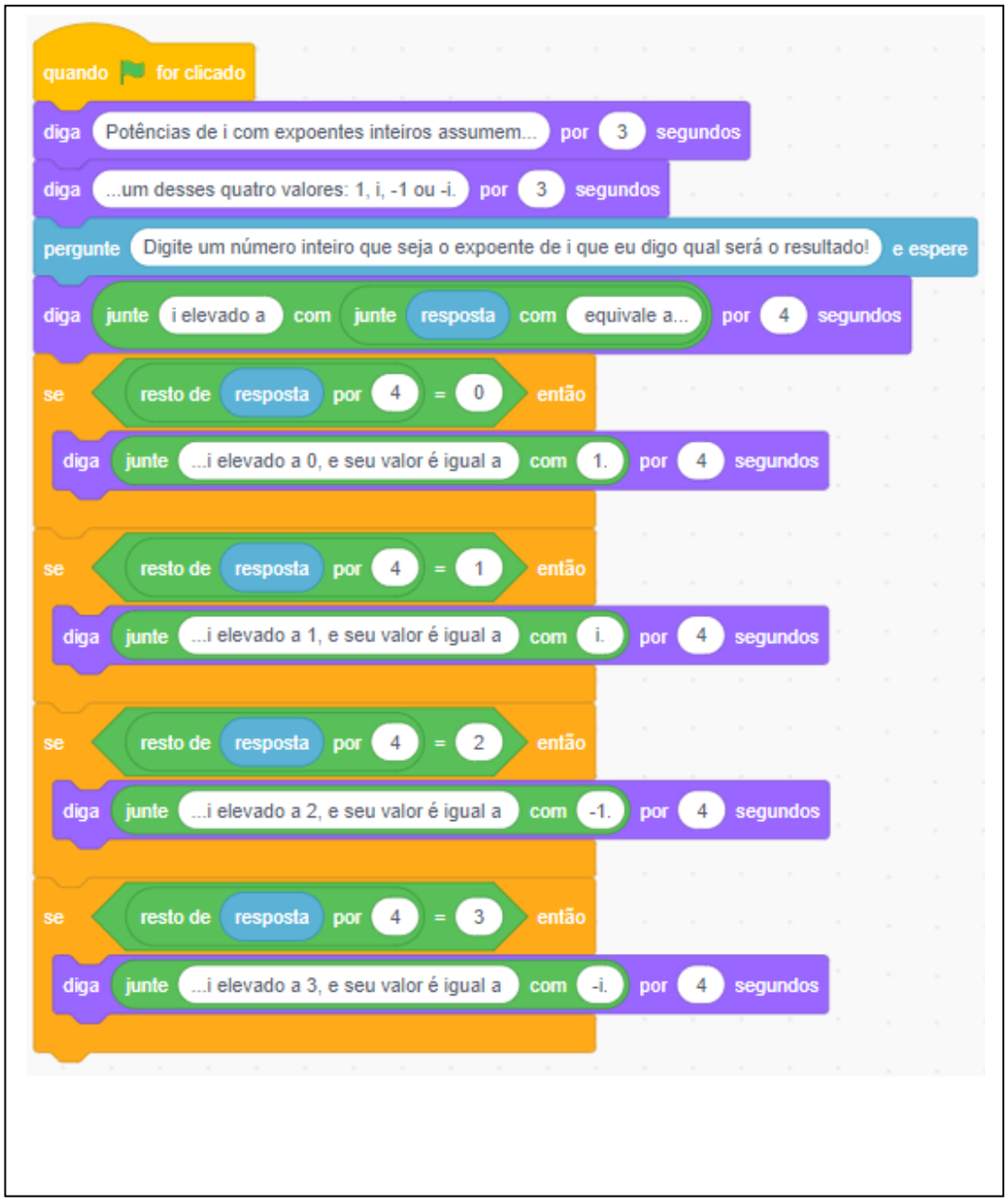
Estabelecer o padrão de comportamento das potências de i (unidade imaginária no conjunto dos números complexos) quando o expoente pertence ao conjunto dos números inteiros. Operar os restos da divisão inteira por 4. De maneira análoga, operar a congruência módulo 4.

3.3.5. Considerações ao professor e sugestão de resoluções

Para essa atividade é importante que os alunos já tenham conhecimento a respeito de números complexos ou acerca da unidade imaginária presente nos números complexos.

É importante destacar que a determinação do resultado final da potência depende de uma avaliação condicional associada ao resto da divisão do expoente por quatro. Dessa forma, a construção do código para resolver o problema necessitará de uma estrutura condicional que avalie o resto da divisão. Uma resolução em *Scratch*[6] é apresentada na Figura 9.

Figura 9 – Potências de i utilizando o *Scratch*



Fonte: elaborada pelo autor (2022)

[6] Disponível em: https://scratch.mit.edu/projects/713999524. Acesso em: 28 dez.2022.

Essa resolução em *Scratch* utilizou quatro testes condicionais de comparação do resto com os possíveis restos da divisão do expoente por quatro. Essa abordagem também poderia ser utilizada na proposta em linguagem C[7], como pode ser visto na Figura 10.

Figura 10 - Potências de i utilizando a linguagem C

```c
#include <stdio.h>
main(){
    int expoente;
    printf(" Calculo de uma potencia de i\n\n");
    printf(" Digite o valor do expoente de i: ");
    scanf("%d",&expoente);
    printf(" O valor de i elevado a %d corresponde a ",expoente);
    if(expoente<0){
        expoente%=4;
        expoente+=4;
    }
    expoente%=4;
    if(expoente==0){
        printf("1");
    }
    if(expoente==1){
        printf("i");
    }
    if(expoente==2){
        printf("-1");
    }
    if(expoente==3){
        printf("-i");
    }
}
```

Fonte: elaborada pelo autor (2022)

Como forma alternativa e introduzindo a utilização de um vetor de caracteres, a Figura 11 apresenta outra maneira de resolver o problema em linguagem C[8].

[7] Disponível em: https://onlinegdb.com/TDnOIIPHS. Acesso em: 28 dez.2022.
[8] Disponível em: https://onlinegdb.com/8pnzjWCHv. Acesso em: 28 dez.2022.

Figura 11 - Potências de i utilizando a linguagem C e vetor de caracteres

```c
#include <stdio.h>

main(){
    int expoente;
    char potencia[4][3] = { {"1"},
                            {"i"},
                            {"-1"},
                            {"-i"} };
    printf(" Calculo de uma potencia de i\n\n");
    printf(" Digite o valor do expoente de i: ");
    scanf("%d",&expoente);
    printf("\n\n O valor de i elevado a %d ",expoente);
    if(expoente<0){
        expoente%=4;
        expoente+=4;
    }
    printf("equivale a %s.",potencia[expoente%4]);
}
```

Fonte: elaborada pelo autor (2022)

Essa proposta utilizou-se de um vetor de cadeias de caracteres (vetor de *strings*) para o armazenamento dos quatro possíveis resultados das potências de i com expoentes inteiros (1, i, –1 e $–i$). Nesse vetor, a posição 0 foi associada ao valor 1, a posição 1 ao valor i, a posição 2 ao valor –1 e a posição 3 ao valor $–i$.

Foi utilizado um teste condicional para ajuste do expoente caso ele seja negativo. Isso se faz necessário pois o operador % (resto ou módulo) na linguagem C retorna valor negativo para expoentes negativos.

A resolução de problemas que requerem a determinação do valor de uma potência de i depende do entendimento de um padrão que ocorre com essas potências. O reconhecimento desse padrão é de fundamental importância para a resolução de qualquer potência de i. Também se faz necessária a abstração, descartando-se o quociente da divisão e tomando-se apenas o resto para a determinação do resultado da potência.

Assim, dois pilares do pensamento computacional têm destaque nessa atividade: a abstração, sendo necessária a identificação da importância do resto da divisão do expoente por 4; e o reconhecimento do padrão de quatro potências existentes que definem todas as demais.

3.4. Atividade 3 – dia da semana

3.4.1. Apresentação

O dia 4 de julho de um certo ano ocorreu em uma sexta-feira. Em qual dia da semana cairá o dia 6 de fevereiro do ano seguinte? Baseado no enunciado anterior, crie um programa para identificar qual será o dia da semana após transcorrido um determinado número de dias a partir de uma data inicial. Esse programa deve pedir em qual dia da semana está uma data inicial e a quantidade de dias que irão se passar até uma data final. O programa deverá informar em qual dia da semana ocorrerá a data final.

3.4.2. Público alvo

Essa atividade pode ser trabalhada com alunos do Ensino Fundamental (séries finais) e com alunos do Ensino Médio.

3.4.3. Duração

Sugere-se um tempo de no mínimo 90 minutos para a aplicação dessa atividade.

3.4.4. Objetivo

Incentivar os alunos na percepção de padrões entre o número de dias transcorridos e o dia da semana. Trabalhar com os possíveis restos da divisão por sete, ou seja, operar congruência módulo 7.

3.4.5. Considerações ao professor e sugestão de resoluções

Nessa atividade é necessário que o aluno perceba que o dia da semana será congruente ao resto da divisão da soma da quantidade de dias com o dia inicial por sete. Pode ser interessante utilizar uma planilha eletrônica, como por exemplo o Excel,

para verificar o dia da semana de uma data inicial específica, bem como para determinar a quantidade de dias entre duas datas fornecidas.

No Excel, a função DIAS(data_final, data_inicial) fornece a quantidade de dias entre duas datas quaisquer. Já a função DIA.DA.SEMANA(data) retorna o código referente ao dia da semana da data requerida. Esse código retornado é um inteiro variando de 1 (representando o Domingo) até 7 (representando o Sábado). Pode-se utilizar o encadeamento de condicionais dentro do Excel para a representação do dia da semana através do seu nome ao invés do código. Se, por exemplo, o código do dia da semana estiver na célula C3, o encadeamento de condicionais seria o apresentado na Figura 12.

Figura 12 - Encadeamento de condicionais para nomear o dia da semana no Excel

```
=SE(C3=1;"Domingo";SE(C3=2;"Segunda";SE(C3=3;"Terça";SE(C3=4;"Quarta";SE(C3=5;"Quinta";SE(C3=6;"Sexta";"Sábado"))))
```

Fonte: elaborada pelo autor (2022)

Uma vez que o programa precisa da data inicial, tanto na abordagem em *Scratch* como em linguagem C, fez-se necessário estabelecer uma forma de entrada dessa data. Foi utilizada uma codificação que associou um código a cada dia da semana. A resolução proposta em *Scratch*[9] pode ser vista na Figura 13.

[9] Disponível em: https://scratch.mit.edu/projects/710898903. Acesso em: 28 dez.2022.

Figura 13 - Determinando o dia da semana utilizando o *Scratch*

Fonte: elaborada pelo autor (2022)

Nessa resolução em *Scratch* foi criada uma lista (Tabela de dias) que é apresentada ao usuário para que este possa digitar o código referente ao dia da semana da data inicial. Essa lista também é utilizada para que o programa exiba o dia da semana após a quantidade de dias digitada pelo usuário.

Nessa proposta foram utilizados códigos de 1 a 7 para identificação dos dias da semana, assim como o Excel utiliza. No entanto, ao final do programa, é necessário verificar se o resto da divisão por 7 é igual a 0. Como a lista tem posições que vão de 1 a 7, caso o resto da divisão por 7 seja 0, o dia correspondente na lista será aquele da posição 7.

A Figura 14 apresenta uma solução para a atividade em linguagem C[10].

Figura 14 - Determinando o dia da semana utilizando a linguagem C

```c
#include <stdio.h>

main(){
    int cont, data_inicio, num_dias;
    char dias[7][15] = { {"sabado"},
                         {"domingo"},
                         {"segunda-feira"},
                         {"terca-feira"},
                         {"quarta-feira"},
                         {"quinta-feira"},
                         {"sexta-feira"},  };
    printf(" Tabela dos codigos dos dias da semana\n\n");
    for(cont=0;cont<7;cont++){
        printf("\t%d\t%s\n",cont,dias[cont]);
    };
    printf("\n Digite a data inicial usando um codigo da tabela: ");
    scanf("%d",&data_inicio);
    printf(" Digite a quantidade de dias: ");
    scanf("%d",&num_dias);
    printf("\n\n Dentro de %d dias sera %s",num_dias,dias[(num_dias+data_inicio)%7]);
}
```

Fonte: elaborada pelo autor (2022)

Nessa resolução foi utilizado um vetor de cadeias de caracteres (vetor de *strings*) para o armazenamento dos nomes dos dias da semana. Nesse vetor, a posição 0 foi associada ao Sábado, a posição 1 ao Domingo, e assim sucessivamente, até a posição 6 associada à Sexta-feira.

Da mesma forma que no *Scratch* utilizou-se a tabela para a entrada da data inicial e para a determinação do dia da semana final, na linguagem C foi utilizado esse vetor de cadeias de caracteres. Assim, a posição do vetor corresponde ao código referente ao dia da semana desejado.

Para a realização dessa atividade, o estudante precisa entender que, dada uma data inicial e uma quantidade de dias, o dia final será determinado pelo reconhecimento do padrão de repetição dos dias da semana. Há uma abstração do

[10] Disponível em: https://onlinegdb.com/om9aJJGF7. Acesso em: 28 dez.2022.

quociente da divisão, importando apenas o resto da divisão por 7 para a determinação do dia da semana para a data final.

Nota-se, nessa atividade, a possibilidade de destaque de três pilares do pensamento computacional. A princípio o pilar da decomposição, uma vez que o problema original tinha uma data inicial e uma data final, o que possibilitou a decomposição em dois problemas menores, um na determinação da quantidade de dias e outro para determinar o dia da semana passados esses dias. Na sequência, o pilar da abstração se mostra presente, uma vez que são descartadas quaisquer outras informações exceto o resto da divisão do número de dias por 7. Finalmente, o pilar do reconhecimento de padrões, que é identificado tanto ao se estabelecer que os dias seguem um padrão de repetição quanto ao se verificar que o problema se assemelha ao que foi proposto no problema anterior.

3.5. Atividade 4 – peças apreendidas

3.5.1. Apresentação

O problema aqui apresentado fez parte do Exame Nacional do Ensino Médio (ENEM)[11] do ano de 2019.

As agências fiscalizadoras divulgam que os cinco principais produtos de autopeças falsificados são: rolamento, pastilha de freio, caixa de direção, catalisador e amortecedor. Após uma grande apreensão, as peças falsas foram cadastradas utilizando-se a codificação:

1: rolamento

2: pastilhas de freio

3: caixa de direção

4: catalisador

5: amortecedor.

Ao final obteve-se a sequência:

5,4,3,2,1,2,3,4,5,4,3,2,1,2,3,4,5,4,3,2,1,2,3,4,…

que apresenta um padrão de formação que consiste na repetição de um bloco de números. Essa sequência descreve a ordem em que os produtos apreendidos foram cadastrados. Determine o 2015º item cadastrado.

Baseando-se no enunciado do problema descrito, elabore um programa que determine qual o item cadastrado na posição de número n. Esse programa deve pedir ao usuário que digite a posição n do item desejado. Na sequência o programa deverá apresentar a peça que se encontra na posição solicitada.

3.5.2. Público alvo

Essa atividade pode ser trabalhada com alunos do Ensino Fundamental (séries finais) e com alunos do Ensino Médio.

[11] Exame Nacional do Ensino Médio (ENEM) – 2019 – 2º dia – Aplicação Regular. Disponível em: https://download.inep.gov.br/educacao_basica/enem/provas/2019/2019_PV_impresso_D2_CD5.pdf

3.5.3. Duração

Sugere-se um tempo de no mínimo 90 minutos para a aplicação dessa atividade.

3.5.4. Objetivo

Reconhecer o padrão fornecido em uma sequência de peças identificadas por números naturais que se encontram em uma determinada ordem. Efetuar a correspondência entre a posição da peça cadastrada e o resto da divisão inteira por oito, ou seja, trabalhar a congruência módulo 8.

3.5.5. Considerações ao professor e sugestão de resoluções

Será essencial, para a resolução do problema proposto nessa atividade, que o aluno perceba a existência de um padrão de oito itens que se repetem na sequência apresentada. Uma vez percebido o padrão de repetição, faz-se necessário estabelecer uma relação entre o resto da divisão da posição do item por oito e a peça correspondente. Dessa forma, haverá uma correspondência entre o resto da divisão e a posição em que o item está armazenado. Para a resolução proposta em *Scratch*, foi utilizada a correspondência apresentada na Tabela 2.

Tabela 2 - Correspondência entre o resto e o respectivo item para abordagem em *Scratch*

Resto	**Posição**	**Peça correspondente**
1	1	amortecedor
0 ou 2	2	catalisador
3 ou 7	3	caixa de direção
4 ou 6	4	pastilhas de freio
5	5	rolamento

Fonte: elaborada pelo autor (2022)

Nessa correspondência, os restos são associados a posições de uma tabela, criada no *Scratch*, contendo os nomes das peças. Quando os restos são valores entre 1 e 5, incluindo ambos, a posição do item na tabela é equivalente ao resto obtido. Se

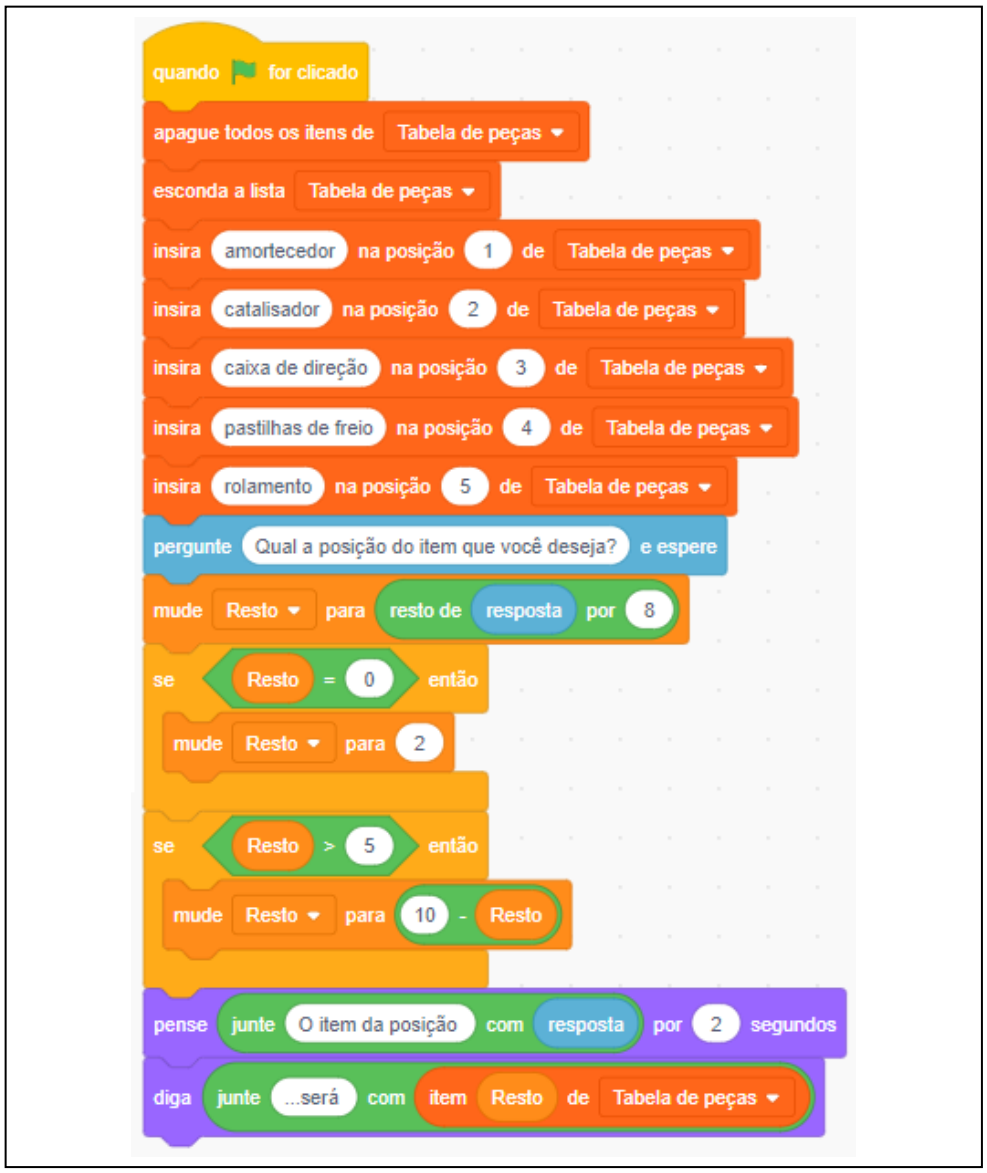
o resto for 0, a posição correspondente na tabela é igual a 2. Para os restos valendo 6 e 7, as posições serão, respectivamente, iguais a 4 e 3.

É possível estabelecer uma função $f: A \to \mathbb{N}$, $A = \{x \in \mathbb{Z} \mid 0 \leq x < 8\}$, em que x representa o resto da divisão da posição desejada por 8 e $f(x)$ apresenta a posição na tabela criada no *Scratch*. Assim:

$$f(x) = \begin{cases} 2 \quad , & x = 0 \\ x \quad , & 0 < x \leq 5 \\ 10 - x, & x > 5 \end{cases}$$

É possível observar essa resolução em *Scratch*[12] na Figura 15.

Figura 15 - Peças apreendidas utilizando o *Scratch*



Fonte: elaborada pelo autor (2022)

[12] Disponível em: https://scratch.mit.edu/projects/715499658. Acesso em: 28 dez.2022.

Para a resolução utilizando a linguagem C, foi criado um vetor com 5 posições; cada posição desse vetor contendo o nome de um dos itens descritos no problema.

Nesse caso, também é necessário estabelecer uma correspondência entre o resto da divisão da posição do item desejado por 8 e a posição do item no vetor. Essa correspondência pode ser verificada na Tabela 3.

Tabela 3 - Correspondência entre o resto e o respectivo item para abordagem em linguagem C

Resto	Posição	Peça correspondente
1	0	amortecedor
0 ou 2	1	catalisador
3 ou 7	2	caixa de direção
4 ou 6	3	pastilhas de freio
5	4	rolamento

Fonte: elaborada pelo autor (2022)

Como as posições de um vetor de tamanho 5 em linguagem C vão de 0 a 4, tem-se uma função $f: A \to \mathbb{N}$, $A = \{x \in \mathbb{Z} \mid 0 \leq x < 8\}$, com x representando o resto da divisão da posição desejada por 8 e $f(x)$ indicando a posição no vetor criado. Logo:

$$f(x) = \begin{cases} 1\ , & x = 0 \\ x - 1, & 0 < x \leq 5 \\ 9 - x, & x > 5 \end{cases}$$

A Figura 16 exibe uma possível resolução do problema em linguagem C[13].

[13] Disponível em: https://onlinegdb.com/T7m6B-vXQ. Acesso em: 28 dez.2022.

Figura 16 - Peças apreendidas utilizando a linguagem C

```c
#include <stdio.h>

main(){
    int posicao;
    char itens[5][20] = {    {"amortecedor"},
                             {"catalisador"},
                             {"caixa de direcao"},
                             {"pastilhas de freio"},
                             {"rolamento"} };
    printf(" Determinacao do item de uma posicao\n\n");
    printf(" Digite a posicao desejada: ");
    scanf("%d",&posicao);
    printf("\n\n O item cadastrado na posicao %d ",posicao);
    posicao%=8;
    if(posicao<1){
        posicao++;
    }
    else{
        if(posicao<6){
            posicao--;
        }
        else{
            posicao=9-posicao;
        }
    }
    printf("corresponde a %s.",itens[posicao]);
}
```

Fonte: elaborada pelo autor (2022)

Esse problema requer que o aluno reconheça um padrão não evidente à primeira vista. Uma vez reconhecido o padrão que possui 8 elementos que se repetem, e não apenas os 5 itens que possuem codificação, é importante estabelecer quais posições correspondem a quais itens.

A abstração, pilar do pensamento computacional, se faz muito necessária nesse sentido, uma vez que era necessário que o estudante ignorasse os 5 itens inicialmente dados em uma classificação e se ativesse à sequência fornecida, composta de 8 itens que se repetem.

Novamente o reconhecimento de padrões é importante, tanto na percepção do padrão de repetição dos 8 itens, quanto na identificação de similaridades com as resoluções obtidas para as duas atividades anteriores (atividades 2 e 3).

3.6. Atividade 5 – tabela numérica

3.6.1. Apresentação

Esse exercício esteve presente no vestibular do ano de 1998 da Universidade Estadual Paulista (Unesp)[14].

Imagine os números inteiros não negativos formando a seguinte tabela:

0	3	6	9	12	...
1	4	7	10	13	...
2	5	8	11	14	...

a) em que linha da tabela se encontra o número 319? Por quê?

b) em que coluna se encontra esse número? Por quê?

Tomando como referência o enunciado desse problema proposto, crie um programa capaz de identificar em qual linha e em qual coluna dessa tabela localiza-se um número inteiro não negativo n. O programa terá como única entrada o valor de um inteiro não negativo n.

Ao final da execução, esse programa deverá informar o número da linha e o número da coluna onde encontra-se o valor digitado n na tabela apresentada pelo problema proposto.

3.6.2. Público alvo

Essa atividade pode ser trabalhada com alunos do Ensino Fundamental (séries finais) e com alunos do Ensino Médio.

3.6.3. Duração

Sugere-se um tempo de no mínimo 90 minutos para a aplicação dessa atividade.

---

[14] Universidade Estadual Paulista – Vestibular 1998 – Por tratar-se de uma prova antiga, não foi encontrada referência no site da Universidade. Disponível em: https://www.curso-objetivo.br/vestibular/resolucao_comentada/unesp/1998/2dia/E_UNESP1998_2dia.pdf

3.6.4. Objetivo

Identificar o padrão na constituição de uma tabela de números inteiros positivos inicialmente fornecida. Operar com o quociente e o resto da divisão inteira por 3.

3.6.5. Considerações ao professor e sugestão de resoluções

Duas abordagens podem ser utilizadas para a elaboração de uma estratégia de pensamento visando a resolução do problema proposto. Em um primeiro momento, pode ser útil imaginar um processo exaustivo que se utilize de um laço de repetição desde o primeiro número da tabela até o número n desejado.

Dentro desse laço de repetição seriam controlados os números de linha e coluna, sendo possível assim determinar a posição de cada número a cada instante. A segunda abordagem consistiria em estabelecer uma relação entre a linha e o resto da divisão de n por 3, e entre a coluna e o quociente da divisão de n por 3. Nessa interpretação de resolução do problema, há a necessidade de que o aluno perceba o padrão que cada número estabelece com sua respectiva posição.

É importante frisar que a primeira abordagem proposta através de um processo exaustivo, por mais que seja uma forma de solucionar o problema, não se traduz como a solução mais eficiente. Quanto maior o número n desejado, tanto maior será o tempo necessário para se obter a linha e a coluna por ele ocupadas.

Assim, a fase de reconhecimento de padrões proposta pelo pensamento computacional – que nesse caso se configura pela disposição dos números constituintes da tabela – permitiria a obtenção de uma resolução mais eficiente, como a segunda proposta apresenta.

A Tabela 4 apresenta os números da tabela fornecida no problema com a identificação do número de cada linha e de cada coluna.

Tabela 4 - Tabela de números com a identificação da linha e coluna ocupadas

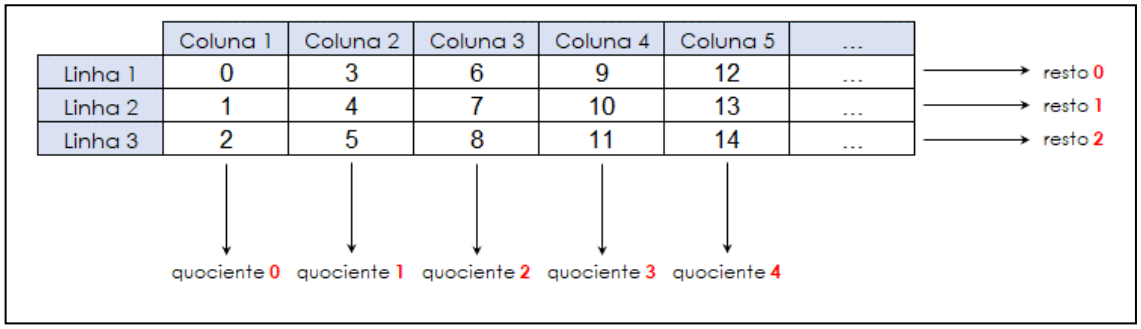
	Coluna 1	Coluna 2	Coluna 3	Coluna 4	Coluna 5	...
Linha 1	0	3	6	9	12	...
Linha 2	1	4	7	10	13	...
Linha 3	2	5	8	11	14	...

Fonte: elaborada pelo autor (2022)

Os números que se encontram na linha 1 têm resto igual a zero quando divididos por 3. Da mesma forma, os números da linha 2 e da linha 3 têm restos iguais a um e dois respectivamente, também quando divididos por 3.

Na coluna 1 encontram-se números que apresentam quociente igual a 0 quando divididos por 3. Na coluna 2 estão os números que apresentam quociente igual a 1 quando divididos por 3. A coluna 3 é constituída pelos números que têm quociente igual a 2 quando divididos por 3. E assim sucessivamente, cada coluna tem um número que excede em uma unidade o quociente dos elementos dessa coluna quando divididos por 3. Os quocientes e restos da divisão de cada número da tabela por 3 são apresentados na Figura 17.

Figura 17 - Tabela de números com a identificação da linha e da coluna ocupadas e indicação do quociente e do resto da divisão por 3

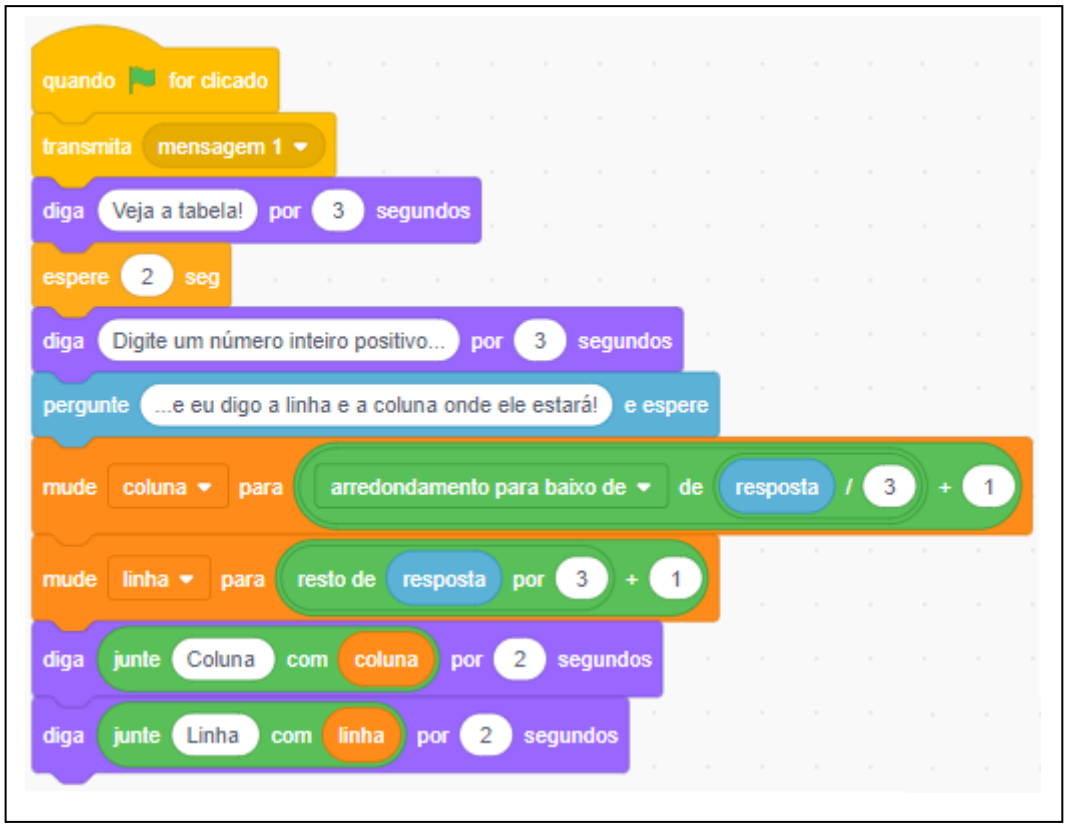


Fonte: elaborada pelo autor (2022)

A solução proposta em *Scratch*[15] está apresentada na Figura 18.

[15] Disponível em: https://scratch.mit.edu/projects/722989838. Acesso em: 28 dez.2022.

Figura 18 - Encontrando linha e coluna ocupadas por um número utilizando o *Scratch*



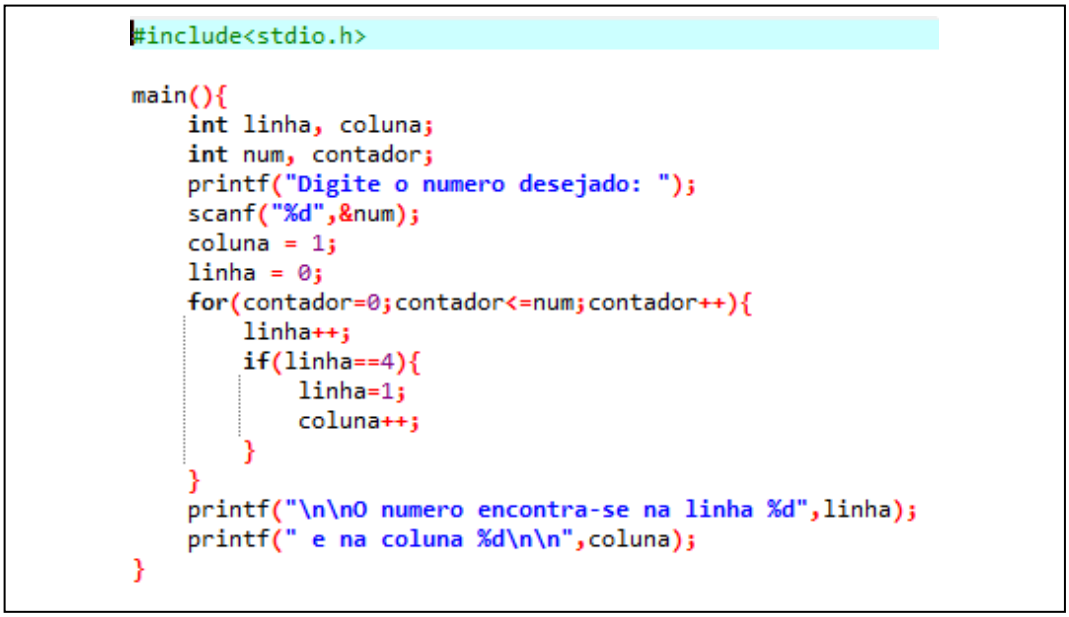
Fonte: elaborada pelo autor (2022)

No *Scratch* faz-se necessária a utilização da função arredondamento para baixo (função piso) para o estabelecimento do quociente inteiro da divisão. Caso esse arredondamento não seja usado, o quociente da divisão pode ser um número real, o que não é desejado nessa análise de resolução.

A Figura 19 apresenta uma solução em linguagem C[16].

[16] Disponível em: https://onlinegdb.com/S6mWjZb7X. Acesso em: 28 dez.2022.

Figura 19 - Encontrando linha e coluna ocupadas por um número através de um método exaustivo utilizando linguagem C



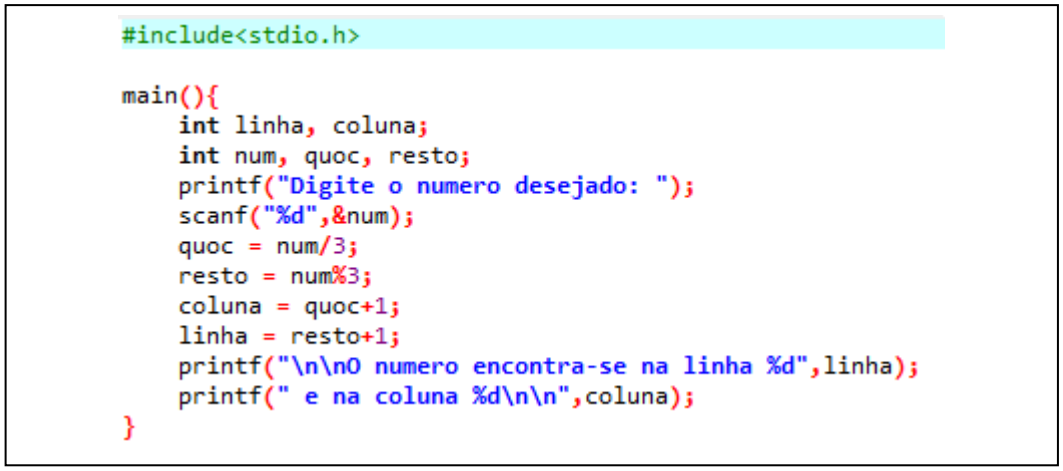
```c
#include<stdio.h>

main(){
    int linha, coluna;
    int num, contador;
    printf("Digite o numero desejado: ");
    scanf("%d",&num);
    coluna = 1;
    linha = 0;
    for(contador=0;contador<=num;contador++){
        linha++;
        if(linha==4){
            linha=1;
            coluna++;
        }
    }
    printf("\n\nO numero encontra-se na linha %d",linha);
    printf(" e na coluna %d\n\n",coluna);
}
```

Fonte: elaborada pelo autor (2022)

Essa abordagem se utiliza de um processo de repetição indo desde o valor zero até o valor n digitado pelo usuário. A cada etapa do laço de repetição são atualizadas as posições de linha e coluna ocupadas pelo número observado naquele momento.

A Figura 20 apresenta outra solução em linguagem C[17].

Figura 20 - Encontrando linha e coluna ocupadas por um número através do quociente e do resto da divisão utilizando linguagem C



```c
#include<stdio.h>

main(){
    int linha, coluna;
    int num, quoc, resto;
    printf("Digite o numero desejado: ");
    scanf("%d",&num);
    quoc = num/3;
    resto = num%3;
    coluna = quoc+1;
    linha = resto+1;
    printf("\n\nO numero encontra-se na linha %d",linha);
    printf(" e na coluna %d\n\n",coluna);
}
```

Fonte: elaborada pelo autor (2022)

[17] Disponível em: https://onlinegdb.com/dO6lkiV-c. Acesso em: 28 dez.2022.

Aqui são utilizados o quociente e o resto da divisão por 3 do número digitado para determinação da coluna e da linha em que ele se encontra.

Esse problema requer uma decomposição em dois problemas menores: um problema de determinação da linha e outro da determinação da coluna. Aqui se evidencia o pilar do pensamento computacional conhecido por decomposição.

O problema de determinação da linha é semelhante aos problemas propostos nas demais atividades desse trabalho. Assim, pode-se esperar novamente pelo reconhecimento de um padrão para a determinação da linha ocupada por um determinado número. Mais uma vez o pilar do pensamento computacional associado ao reconhecimento de padrões se faz presente. Para a determinação da coluna ocupada pelo número, o aluno precisa entender a necessidade da utilização do quociente da divisão. Isso requer um passo extra que não havia surgido nas atividades anteriores.

3.7. Atividade 6 – dígitos verificadores do CPF

3.7.1. Apresentação

O Cadastro de Pessoas Físicas (CPF) é um número de inscrição que contém 11 dígitos numéricos. Usualmente apresenta-se o CPF como na sequência

$$ABC.DEF.GHI - JK$$

onde cada letra corresponde a um algarismo.

Os nove primeiros dígitos são definidos pela Receita Federal no momento da inscrição. Os dois últimos são chamados de dígitos verificadores. Dígitos verificadores são muito comuns para que seja possível identificar erros de digitação. Eles aparecem em números de documentos, agências e contas bancárias, códigos identificadores de livros, entre outros.

Cada dígito verificador do CPF é obtido utilizando-se de 9 algarismos.

O primeiro dígito verificador do CPF (aqui identificado pela letra J) é obtido da seguinte maneira:

1- Calcula-se o valor da somatória, indicada por M, dada por

$$M = 10.A + 9.B + 8.C + 7.D + 6.E + 5.F + 4.G + 3.H + 2.I$$

2- Calcula-se o resto r da divisão de M por 11. Caso r seja igual a 0 ou 1, então o valor de J será 0; caso contrário, o valor de J será igual à diferença $11 - r$.

Para determinação do segundo dígito verificador (identificado pela letra K) é realizada uma operação muito semelhante àquela realizada para o primeiro, seguindo-se os passos:

1- Calcula-se o valor da somatória, indicada por N, dada por

$$N = 10.B + 9.C + 8.D + 7.E + 6.F + 5.G + 4.H + 3.I + 2.J$$

2- Calcula-se o resto s da divisão de N por 11. Caso s seja igual a 0 ou 1, então o valor de K será 0; caso contrário, o valor de K será igual à diferença $11 - s$.

Conhecendo-se o algoritmo de cálculo dos dígitos verificadores do CPF, escreva um programa que solicite ao usuário a digitação dos 9 primeiros algarismos de um CPF. O programa deve calcular e exibir os dois dígitos verificadores.

3.7.2. Público alvo

Essa atividade pode ser trabalhada com alunos do Ensino Fundamental (séries finais) e com alunos do Ensino Médio.

3.7.3. Duração

Sugere-se um tempo de no mínimo 180 minutos para a aplicação dessa atividade.

3.7.4. Objetivo

Efetuar cálculos relacionados aos 9 primeiros dígitos do Cadastro de Pessoas Físicas (CPF), obtendo-se os dois dígitos verificadores. Operar com o resto da divisão inteira por 11.

3.7.5. Considerações ao professor e sugestão de resoluções

Nessa atividade é necessário que o aluno entenda a fórmula proposta para o cálculo de cada dígito, tanto utilizando-se os fatores que multiplicam cada um dos dígitos para a determinação das somatórias, quanto utilizando-se o resto da divisão por 11. Importante considerar também que cada um dos dígitos é determinado fazendo-se um teste condicional. Se o resto da divisão de cada soma por 11 for igual a 0 ou 1, o dígito verificador será igual a 0. Caso o resto seja diferente dos valores 0 e 1, então o dígito verificador será dado por uma diferença entre 11 e o resto obtido.

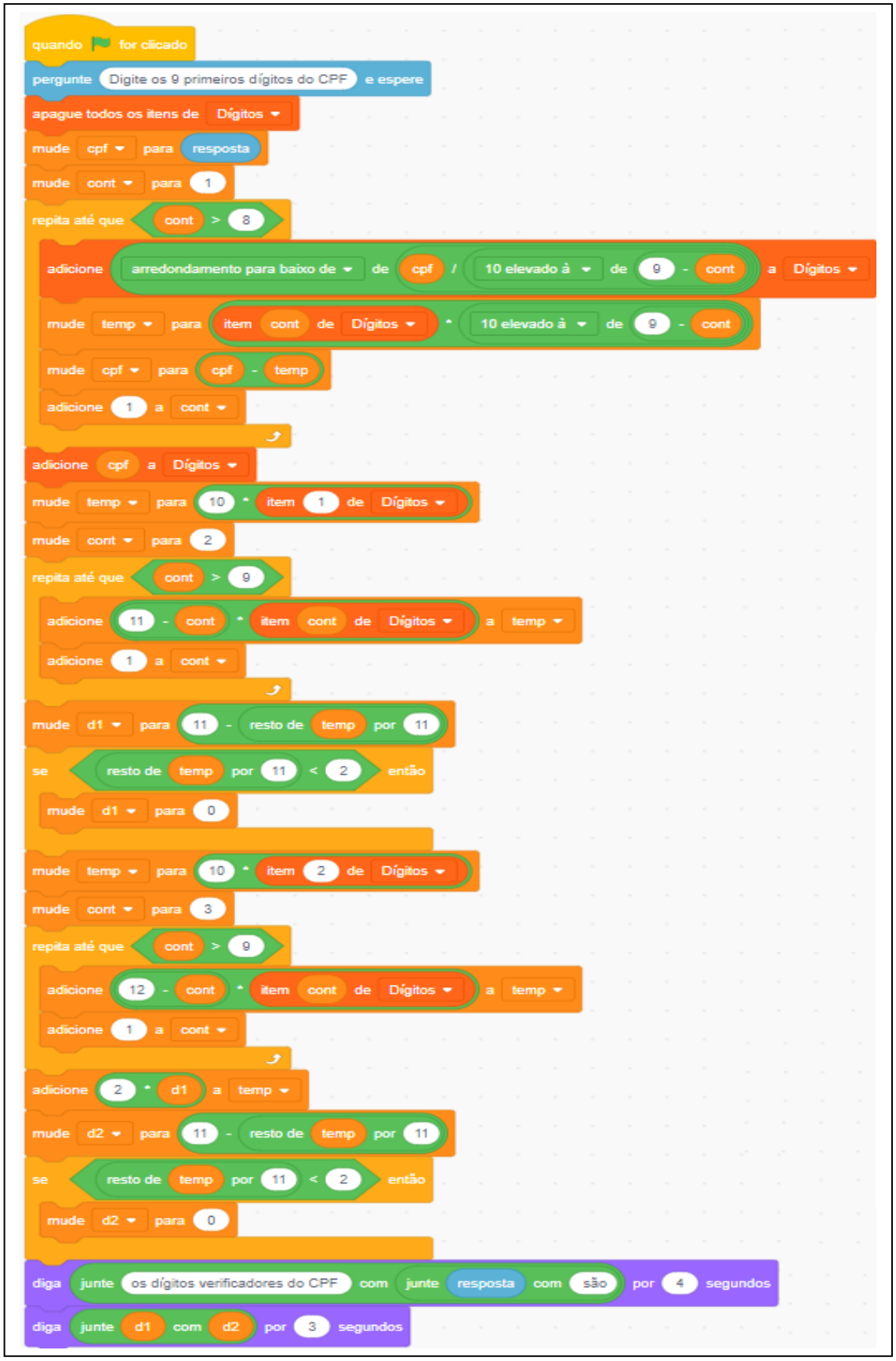
O cálculo dos dígitos é extenso e pode ser de difícil implementação por parte dos alunos. Há a possibilidade de se construir um programa com ou sem[18] a utilização de laços de repetição. Vale destacar que utilizando-se laços de repetição o programa se torna menor, com código mais enxuto e de mais fácil legibilidade. No entanto o uso de laços de repetição pode não ser de fácil entendimento por alunos sem prática de

[18] Para uma resolução sem o uso de laços de repetição em Scratch, há um código disponível na página do autor na plataforma. Disponível em: https://scratch.mit.edu/projects/733601524. Acesso em: 22 nov. 2022.

programação. A Figura 21 apresenta uma proposta de programa em *Scratch*[19] utilizando-se laços de repetição.

[19] Disponível em: https://scratch.mit.edu/projects/767042292. Acesso em: 28 dez.2022.

Figura 21 - Determinação dos dígitos verificadores do CPF em *Scratch*



Fonte: elaborada pelo autor (2022)

Nessa resolução, um laço de repetição é utilizado para que cada dígito dos 9 digitados pelo usuário seja incluído em uma posição de uma lista. Depois, outros dois laços são utilizados, cada um para o cálculo de um dos dígitos verificadores.

A lista criada guarda na primeira posição o primeiro dígito, na segunda posição o segundo dígito, e assim sucessivamente. Logo, a posição do dígito corresponde à posição na lista criada.

Em cada um dos dois últimos laços de repetição foram utilizados os itens da lista de dígitos criada pelo primeiro laço. Para o cálculo do primeiro dígito verificador, cada dígito da lista, que corresponde a cada dígito dos 9 inseridos pelo usuário, será multiplicado por um fator como mostra a Tabela 5.

Tabela 5 - Fatores e posições de dígitos para cálculo do 1º dígito verificador do CPF

Posição do dígito	1ª	2ª	3ª	4ª	5ª	6ª	7ª	8ª	9ª
Fator	10	9	8	7	6	5	4	3	2

Fonte: elaborada pelo autor (2022)

Dessa forma é possível estabelecer que dada uma posição x de um dígito, com $x \in \mathbb{Z}$, $1 \leq x \leq 9$, o fator multiplicativo associado a essa posição será dado por

$$f(x) = 11 - x.$$

De maneira análoga podem-se estabelecer os fatores multiplicativos para a determinação do segundo dígito verificador. Cada laço de repetição foi controlado pela variável cont, que serviu como um contador e indicador da posição dos itens na lista criada. A Figura 22 traz um programa para gerar os dígitos verificadores do CPF na linguagem C[20]. Esse programa também se utiliza de laços de repetição.

[20] Disponível em: https://onlinegdb.com/VI1YkTB11. Acesso em: 28 dez.2022.

Figura 22 - Determinação dos dígitos verificadores do CPF em linguagem C

```c
#include <stdio.h>

main(){
    int cont,d1,d2;
    int temp;
    char cpf[9];
    int digitos[9];
    printf("Digite os 9 primeiros dígitos de um CPF: ");
    scanf("%s",&cpf);
    for(cont=0;cont<9;cont++){
        digitos[cont]=((cpf[cont])-48);
    }
    temp=0;
    for(cont=0;cont<9;cont++){
        temp+=digitos[cont]*(10-cont);
    }
    d1=11-(temp%11);
    if (d1>9){
        d1=0;
    }
    temp=d1*2;
    for(cont=1;cont<9;cont++){
        temp+=digitos[cont]*(11-cont);
    }
    d2=11-(temp%11);
    if (d2>9){
        d2=0;
    }
    printf("Os dígitos verificadores são %i%i",d1,d2);
}
```

Fonte: elaborada pelo autor (2022)

Essa implementação se utiliza de uma cadeia de caracteres (*string*) para a leitura dos 9 dígitos. O primeiro laço de repetição converte cada caractere da cadeia lida em um dígito inteiro de um vetor de inteiros.

Na linguagem C, cada caractere é armazenado como um número na memória. Por exemplo, o caractere ‘A’ é armazenado como número 65, o caractere ‘B’ é armazenado como número 66, e assim por diante. Já o caractere ‘a’ tem número 97, o caractere ‘b’ número 98; ou seja, cada caractere tem um número que o representa.

Os caracteres que representam os algarismos em linguagem C têm os códigos numéricos apresentados na Tabela 6.

Tabela 6 - Códigos numéricos associados aos caracteres
que representam os algarismos em linguagem C

Caractere	0	1	2	3	4	5	6	7	8	9
Código	48	49	50	51	52	53	54	55	56	57

Fonte: elaborada pelo autor (2022)

Assim, dado um caractere com código x, com $x \in \mathbb{Z}$ e $48 \leq x \leq 57$, pode-se determinar o caractere e tratá-lo como um número inteiro correspondente a esse caractere através da função

$$f(x) = 48 - x.$$

Essa foi a operação realizada pelo primeiro laço de repetição. Os outros dois laços se ocupam do cálculo dos dígitos verificadores de forma semelhante ao que foi realizado em *Scratch*.

É importante lembrar que na linguagem C, um vetor ou uma cadeia de caracteres com 9 posições tem essas posições identificadas por índices que variam de 0 a 8. Assim, o primeiro valor estará armazenado na posição 0 do vetor, o segundo valor na posição 1, e assim por diante até que o nono valor estará armazenado na posição 8 do vetor.

Essa atividade propõe uma série de passos para o cálculo dos dígitos verificadores. Para isso ela requer que alguns processos associados ao pensamento computacional sejam realizados. Alguns processos requeridos aqui e também destacados por Barr, Harrison e Conery (2011) são a organização lógica e a análise de dados, além da automatização de soluções através do pensamento algorítmico, caracterizado como uma série de passos ordenados.

Assim, se destaca fortemente o pilar do pensamento computacional conhecido como algoritmos, uma vez que uma sequência muito bem definida de operações deve ser executada para a obtenção dos dígitos desejados.

A decomposição, outro dos pilares do pensamento computacional, se apresenta na necessidade de se dividir o problema em três problemas menores: o primeiro consistindo na leitura dos 9 primeiros dígitos de um CPF e na separação de cada um deles; o segundo e o terceiro correspondendo ao cálculo, respectivamente, do primeiro e do segundo dígitos verificadores.

Já o reconhecimento de padrões pode ser identificado pela necessidade de determinação do resto de uma divisão por 11, mas também pela semelhança dos

passos no cálculo de cada um dos dois dígitos. Novamente faz-se importante frisar que o uso de laços de repetição torna a leitura do código mais simples para quem já apresenta certa prática em programação. Entende-se também que esse seria um passo a ser dado no sentido de obter um aperfeiçoamento da solução, como propõem Palts e Pedaste (2020) em seu modelo cíclico.

Portanto, essa abordagem estaria associada a uma melhora na proposta de uma solução. Isso se relaciona com o que Gal-Ezer e Harel (1998) caracterizam como corretude e eficiência de algoritmos, com o que Selby e Woollard (2013) tratam como a habilidade de pensar em termos de validações, ou mesmo com o que Wing (2006) destaca como resolver um problema de forma eficiente.

4. Considerações finais

Alguns desafios presentes no ensino da matemática, sejam aqueles originados pela falta de interesse dos alunos, sejam os causados pelas dificuldades dos conteúdos trabalhados ou pelas metodologias utilizadas, entre vários outros, são grandes motivadores na busca por novas metodologias. A resolução de problemas, ou uma abordagem que se utilize da investigação matemática, pode ser muito útil no sentido de despertar o interesse dos alunos e mesmo de colocá-los como protagonistas na formação do conhecimento.

Dessa maneira, o pensamento computacional se apresenta como ferramenta de grande importância no processo de ensino e aprendizagem. Como salienta Wing (2006), o pensamento computacional é uma habilidade fundamental para todos e que deve ser incluída como habilidade essencial para todas as crianças, como o são a leitura, a escrita e as operações aritméticas.

Assim, o pensamento computacional se estrutura como habilidade para auxiliar na resolução de problemas, apoiado em quatro pilares destacados por Liukas (2015) como sendo: pensamento lógico e reconhecimento de padrões; pensamento estruturado ou algorítmico; decomposição de um problema; e abstração de um problema.

As atividades propostas nesse trabalho têm a intenção de se utilizar desse aspecto motivador da resolução de problemas, e também de aplicar o pensamento computacional para sistematização dos passos da resolução de problemas. Portanto, o pensamento computacional ganha importância no processo cognitivo. Além disso, acredita-se que o uso de uma linguagem de programação no desenvolvimento dessas atividades melhora, como destacam Garcia, Correia e Shimabukuro (2008), a capacidade de raciocínio lógico e a capacidade de resolução de problemas.

Pode-se concluir que o pensamento computacional em atividades envolvendo a resolução de problemas constitui-se como habilidade de grande relevância que deve ser explorada e ensinada a todos os alunos. Essa habilidade será útil no realce e no fortalecimento de habilidades intelectuais que podem ser transferidas para qualquer área do conhecimento (WING, 2014).

Como propostas de trabalhos futuros destacam-se: a ideia da aplicação dessas atividades em sala de aula e a proposta de um estudo de caso, analisando as

impressões dos alunos no processo de resolução de problemas; a elaboração de outras atividades relacionadas a outros conceitos matemáticos, mas com a mesma abordagem da resolução de problemas com o desenvolvimento do pensamento computacional; e o aprimoramento de técnicas de programação, levando ao conhecimento de diferentes recursos de programação, desenvolvendo habilidades lógicas e possibilitando a resolução de problemas de maior complexidade.

Por fim, espera-se que esse trabalho sirva de incentivo aos inúmeros professores que buscam novas abordagens e novas formas de ensino para melhoria contínua de suas aulas e, consequentemente, para melhoria do aprendizado de seus alunos.

Referências

ABAR, C. A. A. P.; DOS SANTOS, J.; DE ALMEIDA, M. V. Computational Thinking in Elementary School in the Age of Artificial Intelligence: Where is the Teacher?. **Acta Scientiae**. Revista de Ensino de Ciências e Matemática. v. 23, n. 6, p. 270-299, 2021.

ABELSON, H.; SUSSMAN, G. **Structure and Interpretation of Computer Programs**, 2nd ed., Cambridge: MIT Press, 1996.

BARR, D.; HARRISON, J.; CONERY, L. Computational thinking: A digital age skill for everyone. **Learning & Leading with Technology**, 38 (6), p. 20–23, 2011.

BRACKMANN, C. P. **Desenvolvimento Do Pensamento Computacional Através de Atividades Desplugadas na Educação Básica**. 2017. 226 f. Tese (Doutorado) Informática na Educação, CINTED, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2017.

BRASIL. Ministério da Educação. **Base Nacional Comum Curricular**. Brasília, 2018.

COLLING, J. et al. Programação de computadores como meio de desenvolvimento do raciocínio lógico em crianças e adolescentes. **Anais do Seminário de Iniciação Científica do Curso de Pedagogia**, p. 2–8, 2014.

DANTE, L. R. **Matemática em contextos**: função exponencial, função logarítmica e sequências. 1. ed. São Paulo: Ática, 2020.

DEITEL, H. M. **C++: como programar**. 5. ed. Porto Alegre: Pearson, 2006.

FORBELLONE, A. L. V.; EBERSPÄCHER, H. F. **Lógica de programação: a construção de algoritmos e estrutura de dados**. 3. ed. São Paulo: Prentice Hall, 2005.

FRANÇA, R.; TEDESCO, P. Desafios e oportunidades ao ensino do pensamento computacional na educação básica no Brasil. **Anais dos Workshops do Congresso Brasileiro de Informática na Educação**. [S.l.: s.n.], v. 4, n. 1, p. 1464-1473, 2015.

GAL-EZER, J.; HAREL, D. What (else) should CS educators know? **Communications of the ACM**, v. 41, n. 9, p. 77-84, 1998.

GARCIA, R. E.; CORREIA, R. C. M.; SHIMABUKURO, M. H. Ensino de lógica de programação e estruturas de dados para alunos do ensino médio. **XVII WEI–Workshop sobre o Ensino de Computação**. Belém do Pará–PA. [S.l.: s.n.], p. 246–249, 2008.

GARLET, D.; BIGOLIN, N. M.; SILVEIRA, S. R. Ensino de Programação de Computadores na Educação Básica: um estudo de caso. **Revista Eletrônica de Sistemas de Informação e Gestão Tecnológica**, v. 9, n. 2, p. 135-160, 2018.

GUARDA, G. F.; PINTO, S. C. C. S. Dimensões do Pensamento Computacional: conceitos, práticas e novas perspectivas. In*:* SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 31. 2020, Online. **Anais do XXXI Simpósio Brasileiro de Informática na Educação**. Porto Alegre: Sociedade Brasileira de Computação, p. 1463-1472, 2020.

HEFEZ, A. **Aritmética**. Rio de Janeiro: SBM, 2016.

LIUKAS, L. **Hello Ruby**: adventures in coding., New York: Feiwel & Friends, 2015.

LU, J.; FLETCHER, G. Thinking about Computational Thinking. **ACM Sigcse Bulletin**, v. 41, p. 260–264, 2009.

MARQUES, S. G. **Implicação dos pilares do Pensamento Computacional na resolução de problemas na escola**. 2019. 83 f. Dissertação (Mestrado em Educação), Universidade de Santa Cruz do Sul, Santa Cruz do Sul, 2019.

PALTS, T.; PEDASTE, M. A Model for Developing Computational Thinking Skills. **Informatics in Education**, v. 19, n.1, p. 113-128, 2020.

PAPERT, S. **Mindstorms**: Children, Computers, and Powerful Ideas. Basic Books, New York, 1980.

PINTO, A. S. **Scratch na aprendizagem de matemática no 1º Ciclo do Ensino Básico: estudo de caso na resolução de problemas**. 128p. Dissertação (Mestrado em Estudos da Criança – Tecnologias de Informação e Comunicação) - Universidade do Minho, Braga, 2010.

POLYA, G. **How to Solve It**. Princeton: Princeton University Press, 1944.

PONTE, J. P. da; OLIVEIRA, H.; BRUNHEIRA, L.; VARANDAS, J. M.; FERREIRA, C. O trabalho do professor numa aula de investigação matemática. **Quadrante**, [S. l.], v. 7, n. 2, p. 41–70, 1998.

PONTE, J. P. da; BROCARDO, J.; OLIVEIRA, H. **Investigações matemáticas na sala de aula**. 4.ed. Belo Horizonte: Autêntica Editora, 2019.

RESNICK, M. Give P's a chance: Projects, peers, passion, play. **Constructionism and Creativity**: Proceedings of the third international constructionism conference, Viena, 2014. Disponível em: https://web.media.mit.edu/~mres/papers/constructionism-2014.pdf. Acesso em: 20 ago. 2022.

RESNICK, M.; RUSK, N. Coding at a Crossroads. **Communications of the ACM**, v. 63, n. 11, p. 120-127, Nov 2020.

SBC. Sociedade Brasileira de Computação. **Diretrizes para ensino de computação na educação básica**, 2019. Disponível em: https://www.sbc.org.br/documentos-da-sbc/send/203-educacao-basica/1220-bncc-em-itinerario-informativo-computacao-2. Acesso em: 20 out. 2022.

SCHILDT, H. **C: completo e total**. 3. ed. São Paulo: Makron Books, 2006.

SCHMIDT, J.P.; RESNICK, M.; ITO, J. Creative Learning and the Future of Work. In: NORDFORS, D.; CERF, V.; SENGES, M. **Disrupting Unemployment**, Kansas: i4j / Kauffman Foundation, 2016. p. 147-155.

SCRACTH. **Sobre o Scratch**, [s.d.]. Página descritiva. Disponível em: https://scratch.mit.edu/about. Acesso em: 25 abr. 2022.

SELBY, C.; WOOLLARD, J. **Computational thinking: the developing definition**. University of Southampton (E-prints), 2013. Disponível em: https://eprints.soton.ac.uk/356481. Acesso em: 20 ago. 2022.

SETZER, V. W. **Meios eletrônicos e educação**: uma visão alternativa. 2. ed. São Paulo: Escrituras Editora, 2002.

SHUTE, V. J.; SUN, C.; ASBELL-CLARKE, J. Demystifying computational thinking. **Educational Research Review**, v. 22, p. 142-158, 2017.

SOPPELSA, J. J. C. **Divisão euclidiana: um olhar para o resto**. 2016. 226 f. Dissertação (Mestrado em Ensino de Matemática) – Instituto de Matemática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2016.

SOUZA, J. R. **Multiversos Matemática**: Sequências e trigonometria: Ensino Médio. 1. ed. São Paulo: Editora FTD, 2020.

SOUZA, M. F. de; COSTA, C. S. **SCRATCH**: Guia Prático para aplicação na Educação Básica. Rio de Janeiro: Imperial, 2018. Disponível em: https://educapes.capes.gov.br/handle/capes/566023. Acesso em: 11 jul. 2022.

TURKLE, S.; PAPERT, S. Epistemological Pluralism: Styles and Voices within the Computer Culture. **Signs**, v. 16, n. 1, p. 128–57, 1990.

VAN DE WALLE, J. A. **Elementary and Middle School Mathematics**. 4. ed. New York: Longman, 2001.

WING, J. M. Computational Thinking. **Communications of the ACM**, v. 49, n. 3, p. 33–35, 2006.

WING, J. M. Computational Thinking Benefits Society. **Social Issues in Computing**. 2014. Disponível em: http://socialissues.cs.toronto.edu/2014/01/computational-thinking/. Acesso em: 10 abr. 2022.

Apêndice A – Breve apresentação de algumas funcionalidades do *Scratch*

Há uma série de materiais disponíveis para o aprendizado da programação baseada em blocos através do uso do *Scratch*. Inclusive no site de acesso à plataforma do *Scratch*[21] há várias informações para programação de todos os níveis de conhecimento. Uma referência para aplicação na Educação Básica pode ser encontrada em Souza e Costa (2018).

Aqui pretende-se apenas mostrar alguns blocos que foram utilizados nas atividades propostas nesse trabalho. Parte-se do pressuposto de que já foi criada uma conta na plataforma do *Scratch*.

O *Scratch* possui alguns blocos de **Eventos**. Em todas as atividades propostas nesse trabalho foi utilizado apenas um bloco de Eventos para dar início à sequência de comandos. Esse bloco de Eventos associado ao início quando a bandeira verde for clicada pode ser visto na Figura A.1.

Figura A. 1 - Bloco de início quando a bandeira for clicada

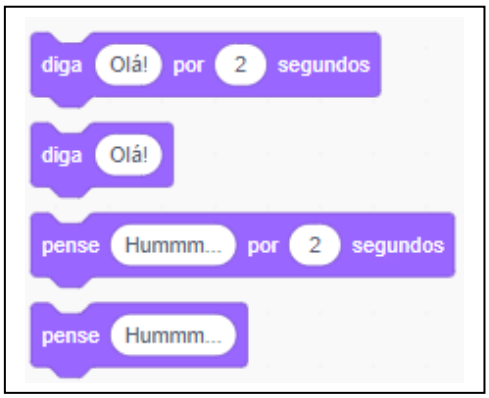
Fonte: elaborada pelo autor (2022)

Quando a bandeira verde é clicada, o *Scratch* entende que deve executar os comandos que estiverem encaixados abaixo desse bloco de Eventos.

Os blocos de **Aparência**, por sua vez, tratam das interações possíveis entre o ator (personagem ou objeto escolhido no projeto) e o usuário. Há blocos de Aparência que são utilizados para exibir mensagens ao usuário, seja na forma de fala do ator, seja na forma de pensamento desse ator, como pode ser visto na Figura A.2.

[21] Página descritiva do *Scratch*. SOBRE o Scratch. Disponível em: https://scratch.mit.edu/about. Acesso em: 25 mai. 2022.

Figura A. 2 - Blocos de aparência de mensagens

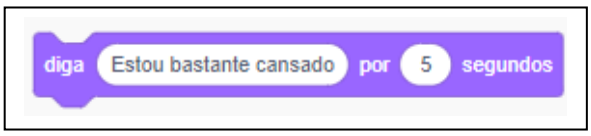


Fonte: elaborada pelo autor (2022)

Tanto os blocos de fala – identificados pela palavra diga – quanto os blocos de pensamento – identificados pela palavra pense – podem ser temporizados. Dessa forma, o ator diz ou pensa algo e a mensagem desaparece após o tempo estabelecido.

Todas as partes que aparecem com fundo branco nesses blocos podem ser alteradas pela mensagem ou pelo tempo desejado. Assim, caso se deseje que o ator diga a mensagem *Estou bastante cansado*, durante um tempo de 5 segundos, basta que se digite isso nos espaços correspondentes, como pode ser visto na Figura A.3.

Figura A. 3 - Bloco com mensagem de exemplo



Fonte: elaborada pelo autor (2022)

Quando executado esse comando, o ator aparecerá como mostrado na Figura A.4.

Figura A. 4 - Mensagem de exemplo dita pelo ator

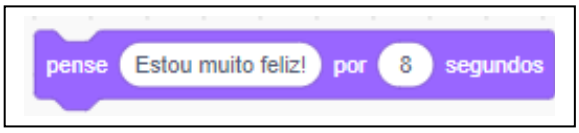
Fonte: elaborada pelo autor (2022)

Ao ser iniciado um projeto no *Scratch*, o ator padrão que aparece é o personagem gato. Pode-se escolher outro personagem ou objeto dentre uma lista de figuras disponíveis no *Scratch*, ou ainda criar seu próprio personagem ou objeto.

Como o interesse nesse trabalho era com o reconhecimento de padrões associado a conceitos de divisibilidade de números inteiros, optou-se por manter o personagem gato em todas as atividades.

Caso seja escolhido o bloco de pensamento, o ator aparece com um balão de pensamento típico das histórias em quadrinhos. Por exemplo, se desejamos que o ator pense a mensagem *Estou muito feliz!*, e que esse pensamento apareça por 8 segundos, o bloco correspondente a essa ação ficaria como mostrado na Figura A.5.

Figura A. 5 - Exemplo de bloco de pensamento



Fonte: elaborada pelo autor (2022)

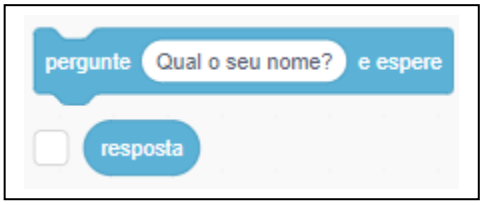
O ator, quando a ação for executada, aparecerá como na Figura A.6.

Figura A. 6 - Exemplo de pensamento do ator

Fonte: elaborada pelo autor (2022)

Dentre a divisão de blocos do *Scratch* chamada **Sensores**, há um bloco de pergunta. Esse bloco, ao ser executado, exibe a mensagem que for digitada nele e espera uma resposta do usuário. A resposta dada pelo usuário fica armazenada e é acessada por um sensor chamado resposta. O bloco de pergunta e o sensor resposta podem ser vistos na Figura A.7.

Figura A. 7 - Bloco de pergunta



Fonte: elaborada pelo autor (2022)

Por padrão a pergunta presente no bloco é *Qual o seu nome?*, e pode ser alterada pela pergunta que se deseje. Basta digitar a pergunta no local com fundo branco do bloco de pergunta.

A resposta pode ser exibida, por exemplo, utilizando-se um bloco de fala combinado com o sensor resposta. Para isso, o sensor resposta deve ser colocado dentro do bloco de fala, como mostra a Figura A.8.

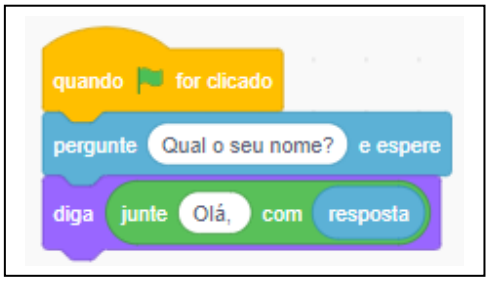
Figura A. 8 - Bloco de fala com o sensor resposta

Fonte: elaborada pelo autor (2022)

Assim, a resposta digitada pelo usuário será apresentada na tela através da fala do ator.

Com esses blocos apresentados até aqui é possível criar um pequeno programa de entrada e saída de dados, fazendo o ator interagir com o usuário. Tomemos por exemplo um programa em que o ator pergunta o nome do usuário. Assim que o nome for digitado pelo usuário, o ator dirá uma frase com o nome que foi digitado. Essa estrutura pode ser vista na Figura A.9.

Figura A. 9 - Exemplo de programa usando blocos pergunta e resposta



Fonte: elaborada pelo autor (2022)

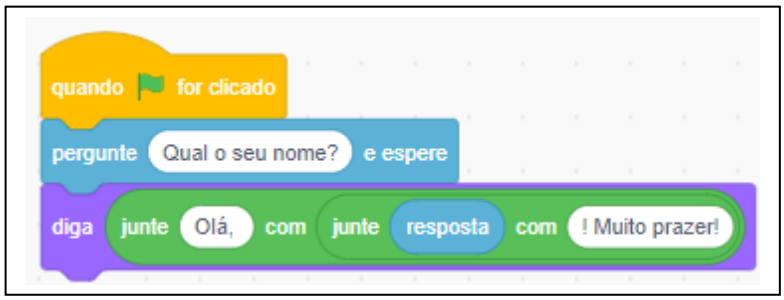
Caso o nome digitado seja João, o ator dirá a mensagem como vista na Figura A.10.

Figura A. 10 - Exemplo de resposta dada pelo ator

Fonte: elaborada pelo autor (2022)

No bloco de fala foi utilizada uma estrutura que aparece no *Scratch* identificada por **Operadores**. Esse operador utilizado serve para juntar duas mensagens. Podem-se utilizar vários desses operadores ao mesmo tempo em uma mesma mensagem. A junção de dois desses operadores pode ser vista na Figura A.11.

Figura A. 11 - Exemplo de uso do operador de junção de mensagens



Fonte: elaborada pelo autor (2022)

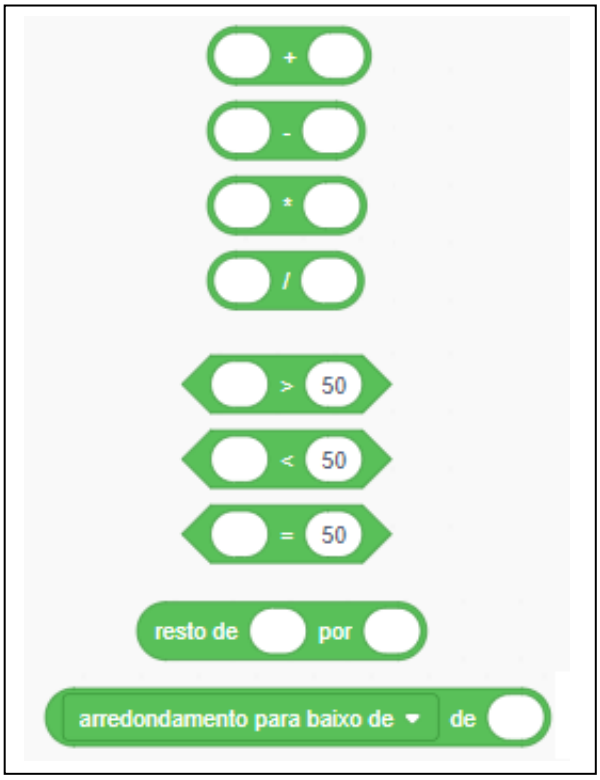
Supondo que na execução desse programa o nome digitado pelo usuário seja Lucas, a mensagem dita pelo ator seria apresentada como na Figura A.12.

Figura A. 12 - Exemplo de fala do ator

Fonte: elaborada pelo autor (2022)

Nas atividades desenvolvidas nesse trabalho, foram utilizados os operadores associados às operações básicas (adição, subtração, multiplicação e divisão), os associados às comparações (maior que, menor que, igual a), um operador associado ao resto da divisão inteira e um operador de arredondamento. Esses operadores são apresentados na Figura A.13.

Figura A. 13 - Blocos de operações básicas, comparações, resto e arredondamento



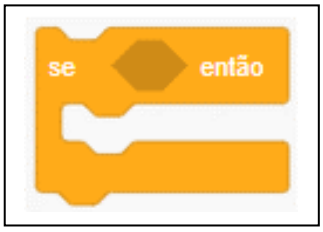
Fonte: elaborada pelo autor (2022)

Assim, é possível combinar resultados de operações ou de comparações para definir alguma execução que se deseje.

Também há uma série de blocos identificados como **Controle**. Com esses blocos é possível trabalhar com blocos de repetição ou com blocos condicionais, controlando o fluxo de execução de determinadas instruções.

Nas atividades realizadas nesse trabalho, utilizou-se somente um tipo de bloco condicional, identificado por se-então, que pode ser visto na Figura A.14.

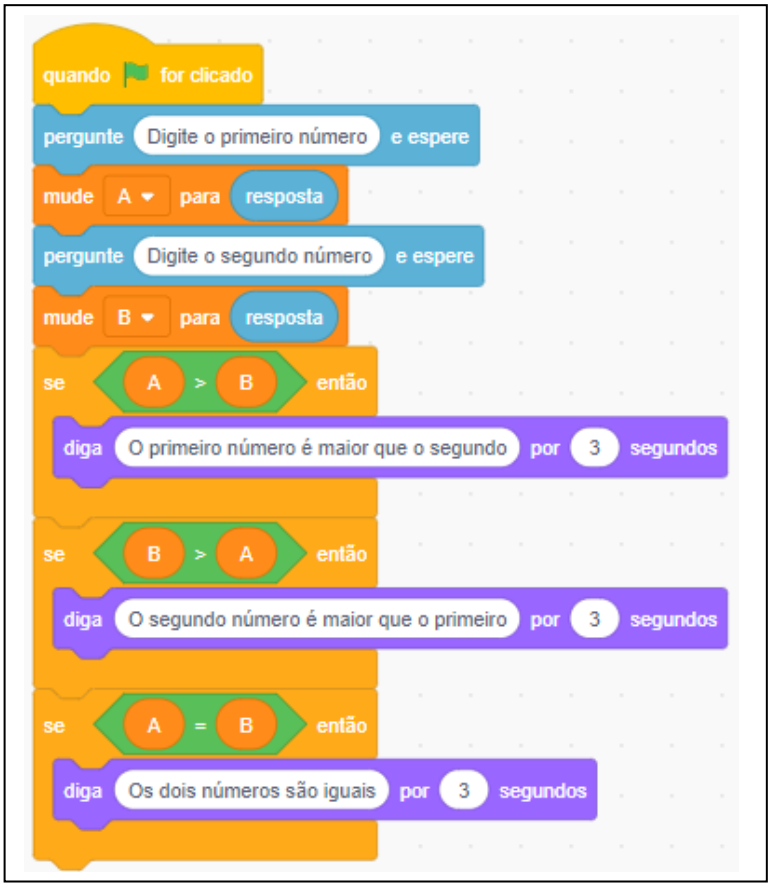
Figura A. 14 - Bloco condicional



Fonte: elaborada pelo autor (2022)

Esse bloco permite a execução de instruções caso a verificação da condição seja satisfeita. Um exemplo de uso desse bloco condicional pode ser visto na Figura A.15.

Figura A. 15 - Exemplo de uso do bloco condicional



Fonte: elaborada pelo autor (2022)

O código presente na Figura A.15 também faz uso de duas variáveis que foram criadas, A e B. Elas foram criadas para armazenarem os dois números que o usuário deverá digitar e para a realização das operações. Nesse exemplo, o ator dirá qual dos dois números digitados é o maior, ou se eles são iguais. As variáveis que forem criadas podem ser manipuladas pelos blocos que aparecem na Figura A.16.

Figura A. 16 - Blocos de manipulação de variáveis

Fonte: elaborada pelo autor (2022)

Com isso é possível alterar o valor de uma variável como se deseje.

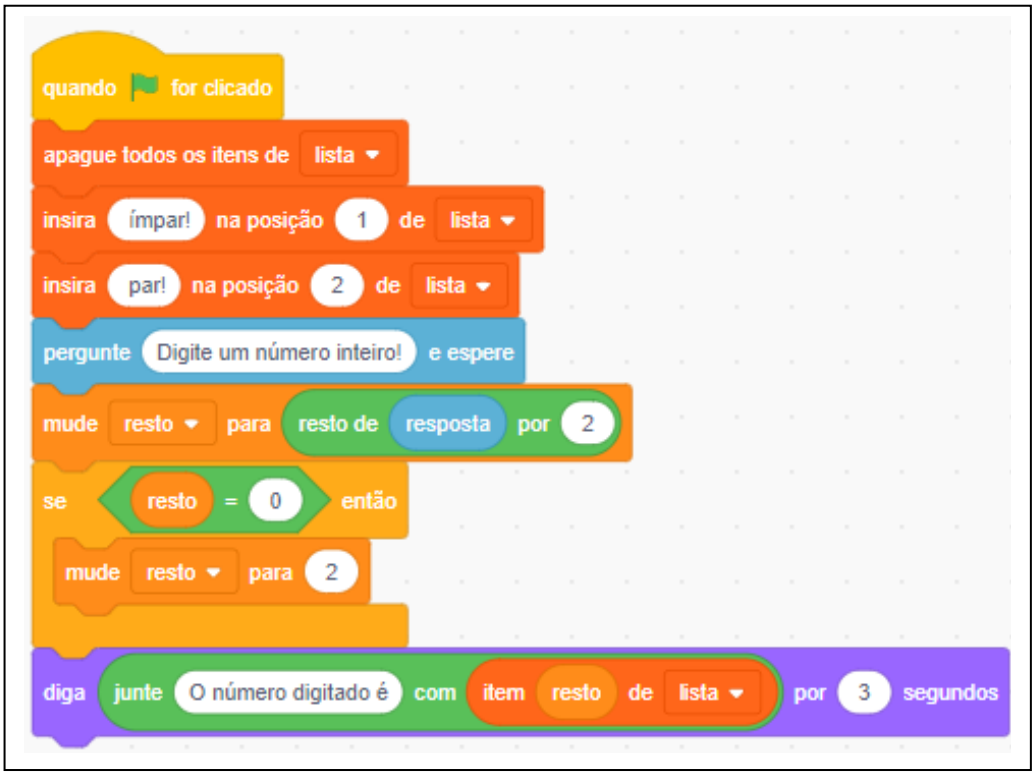
Outra possibilidade para armazenar valores é através da criação de uma lista. A lista funciona na forma de uma tabela ou vetor, em que cada posição pode ser ocupada por algum valor. Alguns dos blocos de manipulação de uma lista podem ser vistos na Figura A.17.

Figura A. 17 - Blocos de manipulação de listas

Fonte: elaborada pelo autor (2022)

Supondo que se queira criar uma lista para dizer se o número digitado pelo usuário é par ou ímpar, isso pode ser feito como mostra a Figura A.18.

Figura A. 18 - Exemplo de manipulação de uma lista

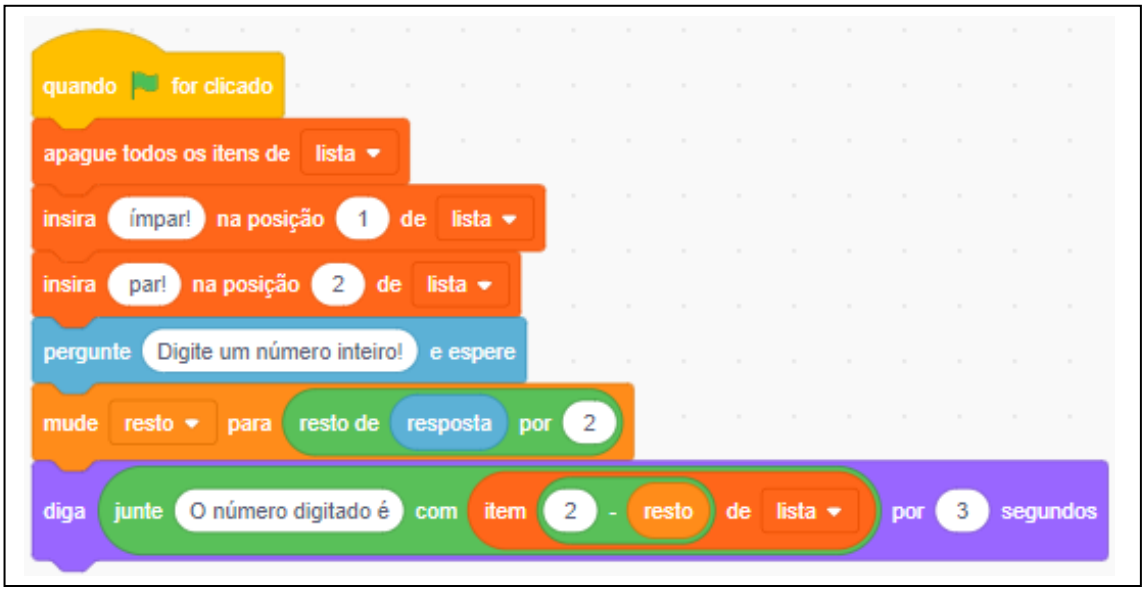


Fonte: elaborada pelo autor (2022)

Nesse caso, utilizou-se a lista para armazenamento das palavras ímpar e par nas posições 1 e 2, respectivamente, da lista criada. Também foi utilizada uma variável para armazenar o resto da divisão do número por 2 e um teste condicional para alterar o valor da variável para 2 caso o resto seja igual a 0.

O mesmo problema pode ser resolvido sem a necessidade de uso da estrutura condicional, como pode ser visto na Figura A.19.

Figura A. 19 - Exemplo de uso de lista



Fonte: elaborada pelo autor (2022)

No entanto, esses são detalhes de implementação cuja discussão cabe para níveis mais avançados de análise de algoritmos. Aqui deseja-se apenas mostrar outra forma de resolver o mesmo problema, o que é bastante comum quando se pensa em algoritmos.

Dentre os blocos identificados como **Controle**, existem blocos de repetição. Estes podem ser utilizados quando uma tarefa ou um conjunto de instruções precisam ser executadas repetidas vezes.

Nas atividades desenvolvidas nesse trabalho utilizou-se apenas um tipo de bloco de repetição. Esse bloco pode ser visto na Figura A.20.

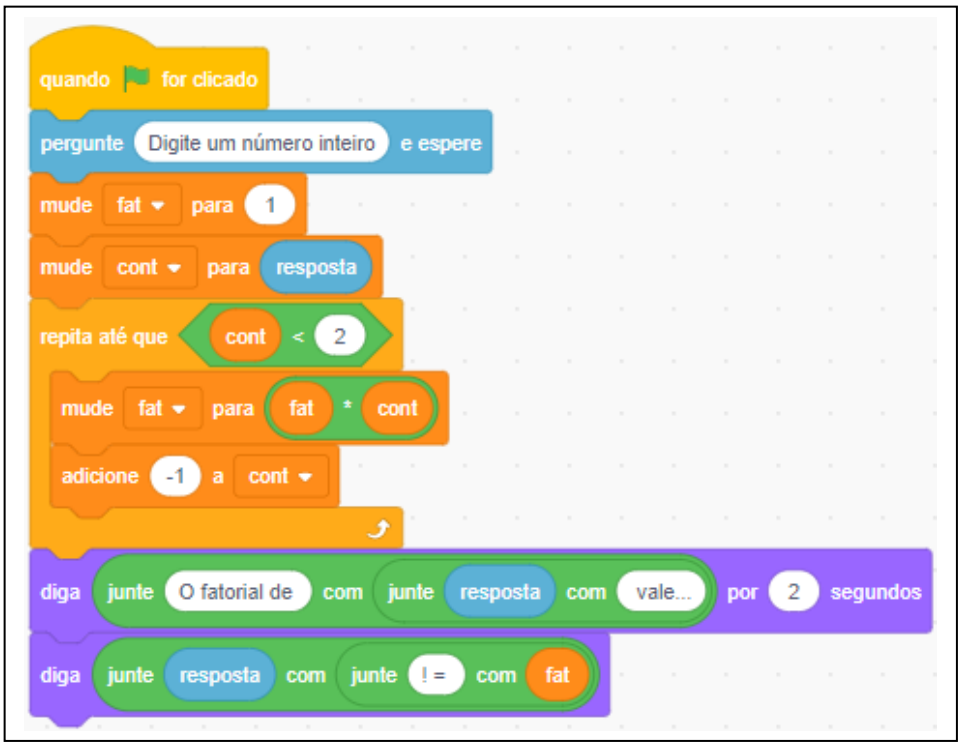
Figura A. 20 - Bloco de repetição

Fonte: elaborada pelo autor (2022)

Esse tipo de bloco exige que um operador de comparação seja utilizado para controlar até quando as instruções dentro dele serão executadas.

Caso se deseje calcular o valor do fatorial de um número digitado pelo usuário, pode-se utilizar um bloco de repetição para efetuar esse cálculo como pode ser visto na Figura A.21.

Figura A. 21 - Exemplo de cálculo do fatorial de um número



Fonte: elaborada pelo autor (2022)

Nesse exemplo, a variável fat é inicializada com o valor 1 e a variável cont, que servirá como uma espécie de contador, recebe o valor digitado pelo usuário. O laço de repetição deverá ser executado até que a variável cont seja menor que 2. Esse, portanto, será o momento em que o laço deixará de ser repetido.

Dentro do laço de repetição, a cada iteração a variável fat é multiplicada pelo valor da variável cont, que vai sendo decrescida a cada execução do laço. Assim, o número vai sendo multiplicado pelos seus antecessores até que se obtenha o valor do fatorial.

Há inúmeras possibilidades de uso de blocos de repetição dentro das mais diversas abordagens e dos mais variados problemas. O objetivo aqui é apenas de apresentar esse bloco como uma possibilidade de auxiliar na resolução de alguns problemas.

Apêndice B – Breve apresentação de algumas estruturas na linguagem C

Existem inúmeras referências para se aprender a respeito da programação em linguagem C. Como essa linguagem foi criada no início dos anos 1970, há livros mais recentes assim como livros com quase meio século. Também há vários vídeos e cursos disponíveis na Internet para o aprendizado da programação em linguagem C.

Uma introdução à linguagem bem como às estruturas de programação disponíveis em C pode ser encontrada em Deitel (2006) ou Schmidt (2006).

A intenção aqui consiste apenas em mostrar algumas instruções que foram utilizadas nas atividades propostas nesse trabalho.

Existem vários compiladores online da linguagem C. Isso permite que o usuário não precise instalar nada em seu computador para escrever um programa ou algoritmo em C e executá-lo. Dois deles são o OnlineGBD[22] e o Programiz[23]. As soluções propostas em linguagem C para as atividades desse trabalho foram compiladas e testadas na plataforma do OnlineGBD.

Para quem deseja instalar um compilador em seu computador, uma recomendação é o Dev-C++[24], um ambiente integrado de desenvolvimento de código aberto para plataformas baseadas no sistema operacional Windows.

Em todas as atividades propostas nesse trabalho, a primeira linha escrita nas resoluções feitas em linguagem C foi a inclusão da biblioteca padrão de entrada e saída. Essa biblioteca é conhecida pelo seu cabeçalho *stdio.h*. A inclusão da biblioteca pode ser vista na Figura B.1.

Figura B. 1 - Inclusão da biblioteca padrão de entrada e saída de dados

```
#include<stdio.h>
```

Fonte: elaborada pelo autor (2022)

[22] Disponível em: https://www.onlinegdb.com/online_c_compiler. Acesso em: 17 jun. 2022.
[23] Disponível em: https://www.programiz.com/c-programming/online-compiler/. Acesso em: 17 jun. 2022.
[24] Disponível em: http://www.bloodshed.net/. Acesso em: 23 abr. 2022.

Dessa biblioteca foram utilizadas duas funções ao longo das atividades propostas. Uma de saída de dados, conhecida como *printf()*, e outra de entrada de dados, *scanf()*.

A função *printf()* exibe na tela a mensagem que se deseje ou o valor de uma ou mais variáveis, ou ainda uma combinação de mensagem com o valor de variáveis. Caso se deseje escrever uma mensagem na tela com a frase *Olá, está tudo bem*, o programa deve trazer as instruções como mostra a Figura B.2.

Figura B. 2 - Exemplo de exibição de mensagem

```
#include <stdio.h>

main(){
    printf("Olá, está tudo bem");
}
```

Fonte: elaborada pelo autor (2022)

Todo programa em linguagem C tem uma função *main()*. Essa função é a função principal do programa e dentro dela virão todas as declarações e funções que serão executadas em C. Assim, pode-se entender que a função *main()* serve para identificar o início e o fim das instruções de um programa em C.

Dentro dos parênteses da função *printf()* deve vir a mensagem que se deseja exibir, entre aspas duplas. A função *printf()* deve ser finalizada com um ponto e vírgula.

A tela, após a execução do programa da Figura B.2, apresentará a mensagem como se vê na Figura B.3.

Figura B. 3 - Mensagem apresentada na tela

```
Olá, está tudo bem
```

Fonte: elaborada pelo autor (2022)

A mensagem escrita entre aspas duplas na função *printf()* pode utilizar controles de exibição. Um dos controles permite avanço de linhas. Esse controle de avanço de linhas é identificado por uma barra invertida seguida pela letra n (\n). Assim, cada vez que aparecer um símbolo do tipo \n, o compilador da linguagem C entenderá que o cursor deve pular uma linha.

Na Figura B.4 é apresentada uma estrutura de programa para exibição da mesma mensagem *Olá, está tudo bem*. No entanto, ela apresenta avanços de linhas.

Figura B. 4 - Exemplo de exibição de mensagem com avanço de linhas

```
#include <stdio.h>

main(){
    printf("\nOlá,\nestá\ntudo\nbem");
}
```

Fonte: elaborada pelo autor (2022)

Na tela, a mensagem aparecerá como mostra a Figura B.5.

Figura B. 5 - Mensagem apresentada na tela em várias linhas

```
Olá,
está
tudo
bem
```

Fonte: elaborada pelo autor (2022)

Caso se deseje que o usuário digite um número inteiro e que o programa apresente esse valor na tela, será necessário criar uma variável para armazenar esse valor lido do teclado. Por se tratar de valor inteiro, deve-se declarar uma variável do tipo inteiro, identificada em C pela palavra reservada int. Ao ser declarada uma variável, é necessário estabelecer um nome para ela e de que tipo se trata. Assim, uma região na memória fica reservada para ser utilizada pelo programa. Suponha que se deseje escrever um programa que peça ao usuário para digitar um número inteiro e que esse número digitado seja exibido na tela. Uma possibilidade de programa que faça o que foi sugerido é apresentada na Figura B.6.

Figura B. 6 - Exemplo de programa para leitura e escrita

```c
#include <stdio.h>

main(){
    int valor;
    printf("Digite um valor inteiro: ");
    scanf("%i",&valor);
    printf("O valor digitado foi %i",valor);
}
```

Fonte: elaborada pelo autor (2022)

A linha de declaração de variável indica que uma variável do tipo inteira está sendo declarada e que essa variável tem o nome valor. Dentro desse programa, em qualquer momento que se queira fazer referência a essa variável, basta utilizar-se o nome valor. As linhas de declarações de variáveis e as linhas de funções sempre terminam com ponto e vírgula.

A função *scanf()* é uma função de entrada de dados pelo teclado. Ela deve indicar o tipo de variável que será lida e o nome da variável onde aquilo que for digitado ficará armazenado. Na instrução utilizada, o tipo de variável é inteiro (identificado pelo código %i ou %d) e a variável tem o nome valor. Para números inteiros na base decimal recomenda-se o uso do código %d. Caso se deseje trabalhar com notações octais ou hexadecimais recomenda-se o uso do código %i. A função *scanf()* precisa do endereço da variável; por isso se faz necessário o uso do símbolo & antes do nome da variável onde o valor será armazenado.

A última função *printf()* faz a junção entre uma mensagem de texto e o valor de uma variável. Dentro da mensagem que encontra-se entre as aspas duplas é necessário informar que tipo de variável será exibida (nesse caso um inteiro, identificado por %i). Logo depois de fechar as aspas duplas, antecedido por uma vírgula, deve ser especificado o nome da variável que se deseja exibir.

Considerando que em uma execução desse programa o usuário digite o número 10, a saída de tela apresentaria o que pode ser visto na Figura B.7.

Figura B. 7 - Tela de execução de entrada e saída

```
Digite um valor inteiro: 10
O valor digitado foi 10
```

Fonte: elaborada pelo autor (2022)

Caso se deseje escrever um programa para ler dois números inteiros do teclado e apresentar a soma deles na tela, pode-se utilizar o código visto na Figura B.8.

Figura B. 8 - Programa para cálculo da soma de dois valores digitados

```c
#include <stdio.h>

main(){
    int a, b;
    printf("Digite o primeiro valor inteiro: ");
    scanf("%i",&a);
    printf("Digite o segundo valor inteiro: ");
    scanf("%i",&b);
    printf("O valor da soma é igual a %i",a+b);
}
```

Fonte: elaborada pelo autor (2022)

Nesse caso, houve a declaração de duas variáveis, chamadas a e b, do tipo inteiro. Pode-se declarar quantas variáveis sejam desejadas, do mesmo tipo, em uma mesma linha, separadas apenas por vírgulas.

A última instrução que utiliza a função *printf()* exibe uma mensagem seguida de um valor inteiro. Nesse caso, logo após a vírgula é apresentada uma soma das variáveis a e b. A linguagem C identifica que o inteiro a ser exibido é o resultado da soma dos valores armazenados nessa variável.

Se durante uma execução desse programa o usuário digitasse 5 para o primeiro valor e 3 para o segundo, o programa apresentaria a tela apresentada na Figura B.9.

Figura B. 9 - Tela exibida pelo programa da soma dos dois números digitados

```
Digite o primeiro valor inteiro: 5
Digite o segundo valor inteiro: 3
O valor da soma é igual a 8
```

Fonte: elaborada pelo autor (2022)

A função *printf()* também permite que uma mensagem contenha valores de várias variáveis ou de resultados de operações entre variáveis. Isso pode ser visto na Figura B.10.

Figura B. 10 - Programa que combina mensagens com variáveis exibidas

```c
#include <stdio.h>

main(){
    int a, b;
    printf("Digite o primeiro valor inteiro: ");
    scanf("%i",&a);
    printf("Digite o segundo valor inteiro: ");
    scanf("%i",&b);
    printf("O valor do produto entre %i e %i é %i",a,b,a*b);
}
```

Fonte: elaborada pelo autor (2022)

A última função *printf()* do exemplo apresenta, dentro da mensagem entre aspas duplas, três identificadores do tipo inteiro (representados por %i). Cada um deles será substituído por uma variável ou operação entre variáveis, na ordem em que aparecem. Assim, o primeiro %i será substituído pela variável a (a primeira a aparecera após a mensagem entre aspas duplas), o segundo %i substituído pela variável b e o terceiro %i pela operação de multiplicação entre a e b (última referência a aparecer).

Um tipo de variável presente na linguagem C é o tipo caractere (identificado pela palavra reservada char). Se uma variável for declarada do tipo caractere, uma posição na memória fica reservada para armazenar uma letra ou símbolo.

Ao longo das atividades propostas nesse trabalho, foram utilizadas cadeias de caracteres. Cadeias de caracteres são como vetores que armazenam sequências de letras ou símbolos.

É possível declarar um vetor de cadeias de caracteres. Assim, pode-se ter em cada posição do vetor uma frase ou uma palavra que se deseje.

Caso se deseje escrever um programa que leia dois números do teclado, compare os dois e diga se o primeiro é maior, igual ou menor que o segundo, pode-se utilizar uma estrutura semelhante ao que aparece na Figura B.11.

Figura B. 11 - Programa de comparação entre dois números

```c
#include <stdio.h>

main(){
    int a, b;
    int comp;
    char relacao[3][12]={
        {"maior que "},
        {"igual a"},
        {"menor que "}
    };
    printf("Digite o primeiro valor inteiro: ");
    scanf("%i",&a);
    printf("Digite o segundo valor inteiro: ");
    scanf("%i",&b);
    if(a>b){
        comp=0;
    }
    if(a==b){
        comp=1;
    }
    if(a<b){
        comp=2;
    }
    printf("O primeiro é %so segundo valor",relacao[comp]);
}
```

Fonte: elaborada pelo autor (2022)

Nesse programa foram executados três testes condicionais (instruções com a palavra reservada if) para determinar qual seria a mensagem a ser exibida. Se o primeiro valor for maior que o segundo, utiliza-se a mensagem da primeira posição do vetor (identificada como posição 0). Se o primeiro e o segundo valores forem iguais, utiliza-se a mensagem da segunda posição do vetor (identificada como posição 1). Por fim, caso o primeiro valor seja menor que o segundo, utiliza-se a mensagem que se encontra na terceira posição do vetor (identificada como posição 2).

As instruções condicionais funcionam da seguinte maneira: um teste é avaliado; caso ele seja verdadeiro, as instruções que estiverem dentro da estrutura daquela condicional serão executadas.

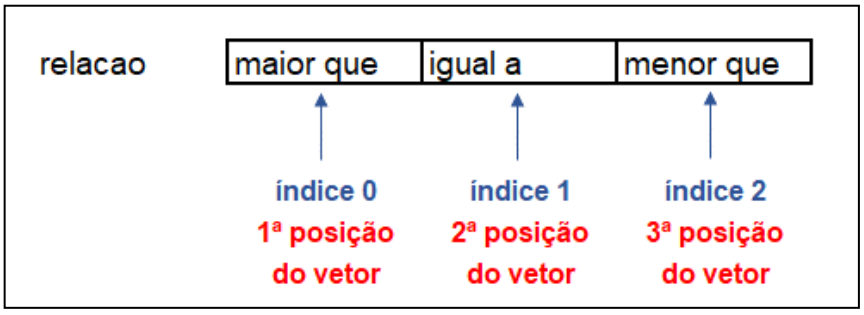
Assim, caso em uma execução desse programa o usuário digite 5 para o primeiro valor e 5 para o segundo, o primeiro teste condicional retornará um valor falso, pois 5 não é maior que 5; o segundo teste condicional retornará um valor verdadeiro, o que fará com que o programa execute a instrução que se encontra dentro dessa condicional, atribuindo o valor 1 à variável comp. Por fim, o terceiro teste condicional também retornará um valor falso, uma vez que 5 não é menor que 5.

Dessa forma será exibida a cadeia de caracteres que se encontra na 2ª posição do vetor (correspondente ao índice 1).

Na função *printf()* é necessário indicar que uma cadeia de caracteres será exibida através do identificador %s.

Para se acessar os dados armazenados em um vetor é necessário fornecer o nome da variável que representa o vetor e a posição específica que se deseja. Nesse exemplo a variável recebeu o nome relacao e cada posição, com sua respectiva mensagem, pode ser vista na Figura B.12.

Figura B. 12 - Posições e índices de um vetor



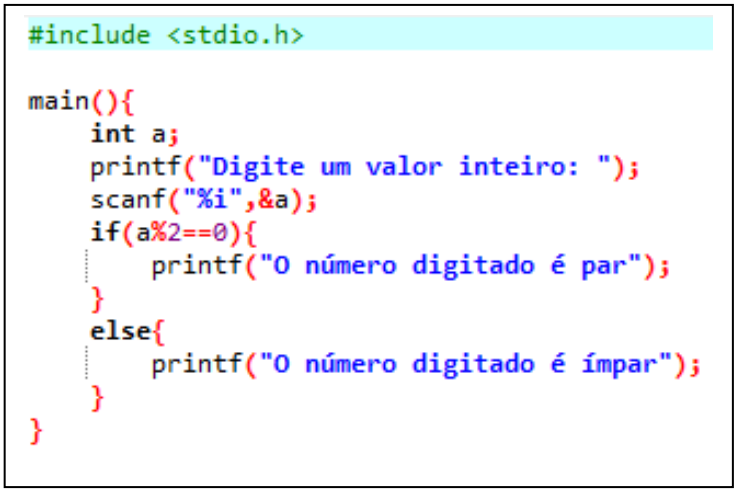
Fonte: elaborada pelo autor (2022)

É importante lembrar que na linguagem C, a primeira posição de um vetor é indicada pelo índice 0, a segunda posição pelo índice 1, e assim sucessivamente.

Outra estrutura condicional que pode ser utilizada é na forma se-senão (estrutura if-else). Nessa estrutura um teste é verificado; caso seja verdadeiro, uma sequência de instruções é executada; caso contrário, outra sequência é executada.

Um exemplo dessa estrutura se-senão pode ser vista na Figura B.13.

Figura B. 13 - Programa que determina paridade de um número usando condicional if-else



```c
#include <stdio.h>

main(){
    int a;
    printf("Digite um valor inteiro: ");
    scanf("%i",&a);
    if(a%2==0){
        printf("O número digitado é par");
    }
    else{
        printf("O número digitado é ímpar");
    }
}
```

Fonte: elaborada pelo autor (2022)

Nesse exemplo, o teste verificado é sobre o resto da divisão do valor armazenado na variável a por 2 (indicado pelo operador %). Caso o resto da divisão por 2 seja igual a 0, então o valor é um número par; caso contrário, ou seja, caso o resto da divisão do número por 2 não seja igual a 0, o valor digitado foi um número ímpar.

Os operadores aritméticos da linguagem C podem ser vistos na Tabela B.1.

Tabela B. 1 - Operadores aritméticos da linguagem C

Símbolo	Função
+	*Adição*
-	*Subtração*
*	*Multiplicação*
/	*Divisão*
%	*Resto da divisão inteira*

Fonte: elaborada pelo autor (2022)

Caso se deseje, por exemplo, atribuir o resultado da divisão da variável a pela variável b a uma variável quociente, a instrução deve conter uma atribuição (representada pelo símbolo =), indicando que a variável quociente recebe o resultado dessa divisão. Isso pode ser visto na Figura B.14.

Figura B. 14 - Atribuindo valor da divisão entre dois valores a uma variável

```c
#include <stdio.h>

main(){
    int a, b, quociente;
    printf("Digite o primeiro valor inteiro: ");
    scanf("%i",&a);
    printf("Digite o segundo valor inteiro: ");
    scanf("%i",&b);
    quociente = a/b;
    printf("O valor da divisão de %i por %i é %i",a,b,quociente);
}
```

Fonte: elaborada pelo autor (2022)

Nesse caso, como a variável quociente é do tipo inteiro, o resultado do quociente da divisão será um valor inteiro. Assim, a linguagem C retorna o resultado do quociente da divisão inteira.

A atribuição de valores a uma variável pode ser combinada com alguma operação envolvendo o valor da própria variável com algum outro valor. Essa combinação pode ser feita com qualquer dos operadores aritméticos, como pode ser visto na Tabela B.2.

Tabela B. 2 - Formas de atribuição de valores a variáveis

Símbolo	Função	Exemplo	Equivalente a
=	atribuição simples	a = b + 3	a recebe a soma de b com 3
+=	atribuição com adição	a+=2	a recebe a soma de a com 2
-=	atribuição com subtração	a-=5	a recebe a diferença entre a e 5
*=	atribuição com multiplicação	a*=2	a recebe o produto de a por 2
/=	atribuição com divisão	a/=7	a recebe a divisão de a por 7
%=	atribuição com resto (módulo)	a%=3	a recebe o resto da divisão de a por 3

Fonte: elaborada pelo autor (2022)

Além desses operadores de atribuição, existem também os operadores de incremento e decremento associados a variáveis. Eles podem ser vistos na Tabela B.3.

Tabela B. 3 - Operadores de incremento e de decremento

Símbolo	Função	Exemplo	Equivalente a
++	Incremento	a++	a recebe a soma de a com 1
--	Decremento	a--	a recebe a diferença entre a e 1

Fonte: elaborada pelo autor (2022)

Os operadores de incremento e decremento podem auxiliar muito em construções em que são necessários laços de repetição.

Suponha que se deseje escrever um programa para calcular o fatorial de um número inteiro n digitado pelo usuário. Como será preciso multiplicar n por seu antecessor $n-1$, que também será multiplicado pelo antecessor $n-2$, até que se obtenha um fator 1, um laço de repetição com uma variável que sofra decrementos a cada vez que o laço esteja ocorrendo torna o algoritmo muito simples. Uma resolução desse problema pode ser vista na Figura B.15.

Figura B. 15 - Cálculo do fatorial de um número utilizando um laço de repetição

```
#include <stdio.h>

main(){
    int n, fat, cont;
    fat=1;
    printf("Digite um número inteiro: ");
    scanf("%i",&n);
    for(cont=n;cont>1;cont--){
        fat*=cont;
    }
    printf("O fatorial de %i vale %i",n,fat);
}
```

Fonte: elaborada pelo autor (2022)

Uma das maneiras de se fazer um laço de repetição na linguagem C é utilizando-se a estrutura for(). Essa estrutura requer em sua sintaxe três partes bem definidas: a inicialização de uma variável, o teste de verificação para definir se o laço será ou não executado novamente e a atualização do valor da variável.

No exemplo sugerido, a inicialização é realizada atribuindo-se à variável cont o valor n digitado pelo usuário. O teste cont>1 define que o laço será executado enquanto a variável cont estiver valendo algo maior do que 1. Por fim, a atualização cont-- faz com que, a cada vez que o laço seja executado, a variável cont seja decrementada. Dentro desse laço, a variável fat vai sendo atualizada a cada momento, sendo multiplicada pelo valor n na primeira vez que o laço é executado, pelo seu antecessor na próxima vez, e assim por diante.