UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE CIÊNCIAS EXATAS E DA TERRA
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ARTURO FONSECA DE SOUZA

SOFTWARE EDUCATIVO DE VISUALIZAÇÃO DE PONTEIROS

NATAL - RN
2025

ARTURO FONSECA DE SOUZA

SOFTWARE EDUCATIVO DE VISUALIZAÇÃO DE PONTEIROS

Monografia de Graduação apresentada ao Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. André Maurício Cunha Campos.

NATAL - RN

2025

Espaço destinado a ficha catalográfica. Deve ser solicitada via SIGAA. Acesse o menu Biblioteca > Serviços ao usuário > Serviços diretos > Ficha catalográfica.

Ao receber a ficha, realize o *download* em PDF/A e anexe ao trabalho com todas as informações contidas na página, inclusive os dados do(a) bibliotecário(a).

ARTURO FONSECA DE SOUZA

SOFTWARE EDUCATIVO DE VISUALIZAÇÃO DE PONTEIROS

Monografia de Graduação apresentada ao Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: 10/07/2025

BANCA EXAMINADORA

Prof. Dr. André Maurício Cunha Campos
Orientador
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Profª. Dra. Anna Giselle Câmara Dantas Ribeiro Rodrigues
Membro interno
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Profª. Dra. Isabel Dillmann Nunes
Membro interno
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Prof. Dr. Selan Rodrigues dos Santos
Membro interno
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Não haveria criatividade sem a curiosidade que nos move e que nos põe pacientemente impacientes diante do mundo que não fizemos, acrescentando a ele algo que fazemos.

Paulo Freire

RESUMO

Na área da computação, ponteiros é um tópico conhecido pela sua dificuldade de compreensão por estudantes. Um dos fortes motivos para isso é a necessidade de, construir na mente do estudante, um modelo mental robusto do computador que descreva a memória e sua relação com as dinâmicas de execução de um programa. Para guiar a construção desses modelos, docentes utilizam, de maneira explícita ou não, modelos abstratos consistentes do computador, os chamados *notional machines*. Este trabalho visa propor uma visualização interativa para essas *notional machines* na forma de um serviço de software fundamentado na Teoria de Aprendizagem Experiencial de Kolb. Usos de caráter exploratório feitos com a ferramenta proposta em turmas ingressantes no curso de Ciência da Computação da UFRN resultaram em impressões majoritariamente positivas por estudantes e evidenciaram pontos de melhoria tanto em aspectos de software como de estratégias pedagógicas.

Palavras-chave: software educativo; ponteiros em c; visualização de programas; aprendizagem experiencial; notional machines.

ABSTRACT

In computer science, pointers are a topic known for being difficult for students to comprehend. One of the main reasons for this is the need to build a robust mental model of the computer in the student's mind that describes the computer memory and its relationship with the dynamics of program execution. To guide the construction of these models, teachers use, either explicitly or not, consistent abstracts models of the computer, known as notional machines. This work aims to propose an interactive visualization for these notional machines in the form of a software service grounded in Kolb's Experiential Learning Theory. Exploratory usages of the proposed tool in introductory Computer Science classes at UFRN resulted mostly in positive impressions from students and highlighted points for improvement in both software aspects and pedagogical strategies.

Keywords: educational software; c pointers; program visualization; experiential learning; notional machines.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE SIGLAS

SVP	Sistemas de Visualização de Programas
SVA	Sistemas de Visualização de Algoritmos
TAE	Teoria da Aprendizagem Experiencial
EC	Experiência Concreta
CA	Conceptualização Abstrata
OR	Observação Reflexiva
EA	Experimentação Ativa
UFRN	Universidade Federal do Rio Grande do Norte
SBC	Sociedade Brasileira de Computação

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

Estudantes de graduação na área de computação precisam aprender diversos conceitos fundamentais relacionados a programação e o funcionamento de computadores no decorrer de seus cursos. O entendimento aprofundado das partes de um computador — especialmente a memória — resulta em um uso mais seguro e eficiente da máquina.

Dentre os conceitos de programação vistos por estudantes nos primeiros anos, um tem sido analisado como particularmente confuso para estudantes: ponteiros (SORVA; KARAVIRTA; MILMA, 2013).

Em linguagens de programação como C e C++, ponteiros são variáveis que possuem como valor o endereço de outro espaço na memória. Esse outro espaço pode conter variáveis de tipos simples ou estrutura de dados mais complexas como *arrays*. No código eles são utilizados como referências indiretas a essas estruturas.

A dificuldade dos colegas de turma do autor deste trabalho com o conceito de ponteiros, em uma das disciplinas introdutórias de programação cursadas por ele, foi o ponto de partida para a elaboração deste trabalho. O contato com visualizações da memória pelo autor fez surgir a hipótese de que essas visualizações seriam uma ferramenta facilitadora do processo de aprendizagem do conceito de ponteiros.

Essa dificuldade, presente em muitas pessoas programadoras iniciantes, pode surgir do fato de que ponteiros exigem um conhecimento mais aprofundado da memória para sua manipulação. Como concluíram Milne e Rowe (2002) em um levantamento sobre a dificuldade de conceitos de programação com professores e estudantes,

> [...] ponteiros e conceitos relacionados à memória [...] só são difíceis pela inabilidade do estudante em compreender o que está acontecendo com seu programa na memória, sendo incapazes de criar um modelo mental claro de sua execução[1] (MILNE; ROWE; 2002, p. 55, tradução própria).

[1] No original: “[...] pointers and memory-related concepts [...] are only hard because of the student’s inability to comprehend what is happening to their program in memory, as they are incapable of creating a clear mental model of its execution”.

Além disso, a introdução de uma nova sintaxe que possui operadores que são excessivamente sobrecarregados, isto é, apresentam diferentes semânticas dependendo de como são utilizados, é a fonte de muitos equívocos por estudantes. Caceffo et al. (2017) observaram diversos problemas de compreensão (*misconceptions*) em estudantes com os operadores associados a ponteiros.

Em um levantamento realizado por Arimoto e Oliveira (2019) com centenas de estudantes de cursos técnicos e superiores na área de computação no Brasil foi observado que os conceitos de ponteiros e recursão foram um dos mais avaliados como “muito difícil”. O mesmo resultado foi obtido por um levantamento internacional com professores e estudantes feito por Lahtinen, Ala-Mutka e Järvinen (2005).

Um estudo mais recente, elaborado por Pereira e Sousa (2024) na Universidade Federal Rural do Semi-Árido (UFERSA), Campus de Pau dos Ferros - RN, indicou que mais de 80% dos estudantes participantes consideram o tema de ponteiros como sendo o conceito de programação que apresentaram maior dificuldade no seu aprendizado.

Logo, é perceptível a permanência desse obstáculo conceitual na vida acadêmica de estudantes iniciantes de computação ao longo das décadas e em diferentes partes do mundo.

Este trabalho busca apresentar uma solução que ajude a reduzir essa barreira no processo de aprendizagem desses estudantes, se apoiando na teoria de modelos mentais, aprendizagem experiencial e no conceito de *notional machines*.

1.2 Objetivos

Este trabalho tem como objetivo geral facilitar o processo de aprendizagem do conceito de ponteiros por parte de estudantes iniciantes através de um sistema de visualização de programas. De forma mais específica, este trabalho procura alcançar os seguintes objetivos:

- Desenvolver uma solução computacional em que o usuário seja capaz de visualizar uma representação visual das dinâmicas de execução de um programa com foco em ponteiros;
- Permitir que a ferramenta seja adequada tanto para uso por estudantes quanto por professores em aulas expositivas;
- Avaliar impressões dessa solução por parte dos estudantes.

1.3 Justificativa

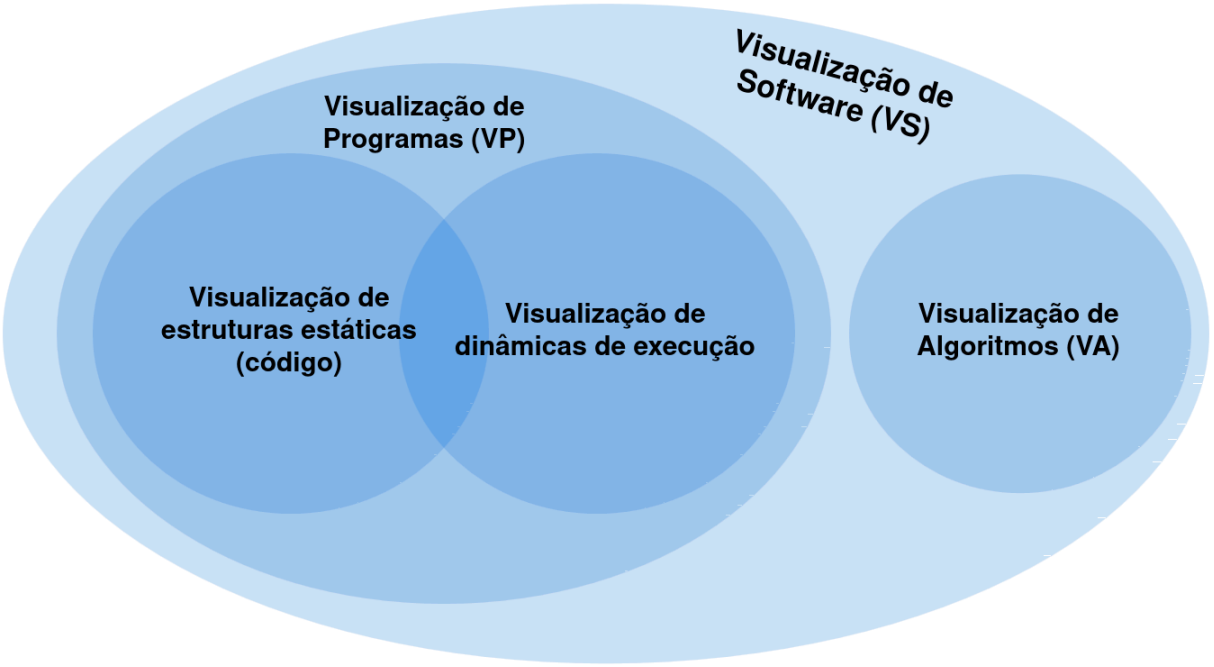
Uma pessoa, ao programar, não costuma pensar nos detalhes específicos da máquina que executará seu código. Características da memória física ou os circuitos lógicos que compõem um processador são muitas vezes abstraídos em um modelo mental da máquina. Um nível de abstração adequado lhe permite utilizar, de forma eficiente, as ferramentas fornecidas pela linguagem de programação escolhida.

Estudantes iniciantes possuem um modelo mental do computador-linguagem muitas vezes inconsistente e errôneo, ao passo que pessoas programadoras mais experientes possuem um modelo mais próximo do ideal. Este modelo ideal é chamado de "*notional machine*", descrito pela primeira vez por du Boulay et al. em 1981 (SORVA; KARAVIRTA; MILMA, 2013). Esses modelos abstraem, ao nível apropriado, os detalhes do funcionamento de uma máquina no contexto de uma linguagem de programação, de forma consistente e correta (DICKSON; BROWN; BECKER, 2020).

Existem muitas formas de visualizar *notional machines*. Essas visualizações buscam mostrar o estado interno do computador durante a execução de um programa. Elas podem ser desenhadas em lousas de sala de aula ou de forma mais dinâmica com ferramentas de software. Diversos sistemas de classificação foram desenvolvidos para classificar softwares de visualização. A classificação que este trabalho utilizará é baseada no artigo de Sorva, Karavirta e Milma (2013) e pode ser visualizada na Figura 1.

Nessa classificação, há dois grupos principais: Sistemas de Visualização de Programas (SVPs) e Sistemas de Visualização de Algorítmos (SVAs). O último se refere a sistemas que visualizam algoritmos (como *quicksort* e busca binária) que possuem geralmente um alto nível de abstração. Este tipo não será discutido neste trabalho.

Figura 1 – Classificações de visualizações de software



Fonte: Adaptado de Sorva, Karavirta e Milma (2013).

Já SVPs tipicamente apresentam visualizações com um menor nível de abstração. Eles representam tanto estruturas estáticas presentes no código como de ações realizadas em tempo de execução. Auxiliam no entendimento de diversos conceitos de programação, sejam laços, entradas de funções ou avaliação de expressões. O seu papel, como de outros sistemas de visualização de software, é aproximar ao máximo o modelo mental que o estudante tem da linguagem e do computador com o *notional machine* representado pelo SVP.

Este trabalho apresenta um SVP em forma de um *website* que auxilia estudantes a entender conceitos de gerenciamento de memória, especialmente ponteiros, cobrindo as limitações de sistemas já existentes.

A escolha de um SVP como ferramenta pedagógica é posta em contraste contra métodos expositivos tradicionais de visualização, isto é, desenhos feitos em quadros de aula. Há a possibilidade de utilizá-lo em aulas expositivas, como material auxiliar de slides ou livros didáticos digitais e também como ferramenta interativa de acesso direto por estudantes.

Ao interagir com a ferramenta o estudante passa a ser um agente ativo na construção de seu conhecimento. Esse processo é legitimado pela Teoria de

Aprendizagem Experiencial de Kolb (2015), que será discutida com profundidade na seção 2.

1.4 Estrutura do documento

Na seção 2 é discutido os dois principais arcabouços teóricos deste trabalho: a Teoria de Aprendizagem Experiencial de Kolb que legitima a escolha de um software de visualização interativo por uma ótica construtivista e as *notional machines* como ferramentas pedagógicas associadas profundamente as visualizações que as representam e aos modelos mentais dos estudantes.

Na seção 3 são apresentados trabalhos relacionados, muitos dos quais são listados na revisão feita por Sorva, Karavirta e Milma (2013) sobre SVPs.

Na seção 4 a solução proposta é apresentada como sendo dividida em duas partes: *backend* e *frontend*. Detalhes de como o código do usuário é compilado, executado em um ambiente seguro e transformado em um formato que possa ser utilizado pela visualização são expostos junto às funcionalidades presentes na ferramenta.

Na seção 5 é discutida a metodologia utilizada e os usos da ferramenta de caráter exploratório feitos em salas de aula.

Na seção 6 os resultados são apresentados e discutidos. As impressões de estudantes junto aos resultados de seus testes são analisadas em conjunto para entender os seus problemas de compreensão e se a ferramenta proposta é capaz de auxiliar nesses processos de aprendizagem.

Na seção 7 uma discussão mais aprofundada sobre os possíveis caminhos que a ferramenta pode tomar com base nas impressões e sugestões de estudantes é realizada.

Na seção 8 conclusões são formuladas acerca do uso da solução proposta em sala de aula e as impressões geradas. Sugestões são feitas para trabalhos futuros focando na diversidade de pesquisas e na melhoria da ferramenta.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Aprendizagem experiencial

"É praticamente impossível encontrar um pesquisador em educação atual que acredite que a aprendizagem envolve simplesmente a transmissão, ou 'derramamento' de conhecimento pré-existente de um livro ou professor(a) em estudantes"[2] (SORVA, 2013, p. 13, tradução própria). Com essas palavras, Sorva (2013) quis dizer que, hoje em dia, praticamente "somos todos construtivistas".

O construtivismo é uma teoria educacional que afirma que o conhecimento é construído ativamente pelo indivíduo, em oposição a paradigmas clássicos que focam no recebimento e armazenamento do conhecimento transmitido por professores ou absorvidos através de livros. Freire (2018) formula o conceito de "educação bancária" para descrever tais modelos de aprendizado:

> Desta maneira, a educação se torna um ato de depositar, em que os educandos são os depositários e o educador o depositante. Em lugar de comunicar-se, o educador faz "comunicados" e depósitos que os educandos, meras incidências, recebem pacientemente, memorizam e repetem. Eis aí a concepção "bancária" da educação, em que a única margem de ação que se oferece aos educandos é a de receberem os depósitos, guardá-los e arquivá-los. [...] Só existe saber na invenção, na reinvenção, na busca inquieta, impaciente, permanente, que os homens fazem no mundo, com o mundo e com os outros. Busca esperançosa também. (FREIRE, 2018, p. 80).

Paulo Freire é um dos pesquisadores que integram o grupo de principais fundadores da Teoria de Aprendizagem Experiencial (TAE) de Kolb (2015). Em seu livro homônimo, publicado originalmente em 1984, Kolb descreve um modelo de aprendizagem em que "a aprendizagem é o processo pelo qual o conhecimento é criado através da transformação da experiência"[3] (KOLB, 2015, p. 49, tradução própria). É nesta teoria em que a solução proposta por este trabalho se legitima como meio viável de mediação do aprendizado.

---

[2] No original: "It is nigh on impossible to find a present-day educational researcher who believes that learning simply involves the transmission, or "pouring," of pre-existing knowledge from a teacher or a book into students".

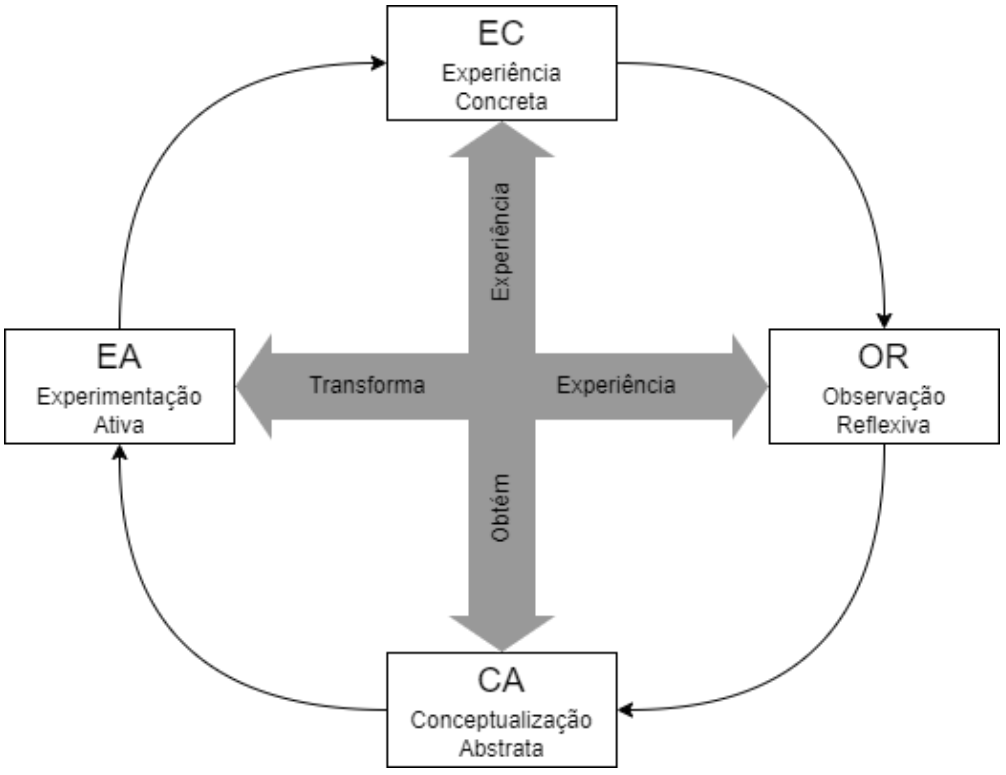
[3] No original: "Learning is the process whereby knowledge is created through the transformation of experience".

Kolb (2015) enfatiza que a mente do estudante não é uma “tábula rasa” e que todas as pessoas enfrentam um novo processo de aprendizagem com conceitos prévios. Esse processo de contínua modificação e integração de conhecimentos com foco na subjetividade do consciente se contrapõe à “aprendizagem por experiência” descrita pela abordagem behaviorista que dominou os estudos de aprendizado na primeira metade do século XX. Nela, o processo subjetivo de processamento de experiências que ocorre na mente é praticamente ignorado em detrimento de observações comportamentais que comprovam o aprendizado.

Para Kolb (2015), a aprendizagem pode ser separada nas formas de obter experiência e nas formas de transformá-la. As formas de obter experiência são definidas por dois pólos relacionados dialeticamente: Experiência Concreta (EC) e Conceptualização Abstrata (CA). Já as formas de transformar a experiência são definidas por Observação Reflexiva (OR) e Experimentação Ativa (EA).

Esses 4 modos de aprendizagem compõem algo central na teoria de aprendizagem de Kolb: o Ciclo de Aprendizagem (Figura 2). É através do percorrimento entre eles que o processo de aprendizagem se forma (KOLB, 2015).

Figura 2 – Ciclo de aprendizagem



Fonte: Adaptado de Kolb (2015).

A solução proposta por este trabalho (detalhada na seção 4) é composta, essencialmente, de duas partes: uma onde é possível escrever um programa nas linguagens de programação C ou C++ e outra que exibe uma representação da memória de um computador abstrato.

É esperado que, ao interagir com a solução proposta, a pessoa estudante perpasse de maneira quase instantânea por todos estágios do Ciclo de Aprendizagem de Kolb: ao escrever o código (EC), ao refletir sobre a representação alterada (OR), ao pensar o porquê da alteração (CA) e ao voltar a escrever com um novo olhar (EA).

O computador, por ser um sistema causal, implica que o estágio de Conceptualização Abstrata também representa a formação de modelos mentais que consigam organizar as observações feitas, processo também chamado de "teorização" por Kolb (2015). A ferramenta pedagógica que pode ajudar a guiar esse modelos mentais , tornando-os menos suscetíveis a equívocos e inconsistências, é discutida na próxima seção secundária.

Como afirmam Sorva, Karavirta e Milma (2013), multimídias interativas que implicam em um engajamento cognitivamente ativo por parte do usuário resultam em um processamento mais elaborativo que se integra com conhecimentos prévios. Outro benefício identificado é que estudantes acabam passando mais tempo estudando ao terem contato com essas ferramentas, intensificando o processo de aprendizagem.

Desta forma, no processo de transformação de experiência na interação dessa ferramenta, o conhecimento da pessoa estudante acerca do tópico de interesse poderá ser mais facilmente construído do que por meios mais tradicionais, tais como aulas expositivas ou mídias passivas.

2.2 *Notional machines* e modelos mentais

Pessoas estabelecem modelos mentais ao interagir com sistemas dos mais diferentes tipos. São estruturas que podem ser utilizadas para descrever o propósito e mecanismos internos de sistemas para si. No caso de sistemas causais como circuitos e *software*, esses modelos ajudam também a prever estados futuros do sistema (SORVA, 2013).

Uma pessoa, ao programar, forma um modelo mental acerca do funcionamento do computador e das dinâmicas de execução do programa ao escrevê-lo. Esses modelos muitas vezes são incompletos e repletos de equívocos em estudantes iniciantes. Pela ótica do paradigma construtivista, seus conhecimentos não são ditos “errados”, mas que acabam não sendo viáveis no contexto específico (BEN-ARI, 1998).

Tanto Sorva (2013) como Ben-Ari (1998) defendem que o computador possui tão pouca margem para interpretações que divergem de seu modelo normativo que, ao contrário de visões construtivistas mais radicais, apresenta uma realidade ontológica conectada à realidade. Ou seja, ele possui um conjunto de conhecimentos que podem ser considerados “certos” dentro de seu contexto, o que diverge da epistemologia construtivista. Além disso, a natureza do computador ao executar programas torna essa realidade ontológica acessível (BEN-ARI, 1998), no sentido de que programas desenvolvidos a partir de conhecimentos que não estejam bem alinhados com sua realidade prontamente apresentam isso à pessoa programadora na forma de mensagens de erro, travamentos e *bugs*.

O trabalho de muitos docentes na área da computação é fazer com que os modelos mentais de estudantes se aproximem o máximo possível de um modelo ideal que é consistente e robusto o suficiente para explicar todos os fenômenos da máquina real que ele abstrai no contexto da linguagem sendo utilizada. A este modelo ideal foi denominado de “*notional machine”*.

O conceito de *notional machine* foi criado por du Boulay em 1981, buscando formalizar abstrações utilizadas no contexto do ensino de programação para iniciantes (SORVA, 2013). Nas palavras de du Boulay, O’Shea e Monk, é “um computador conceitual cujas propriedades estão implícitas nas estruturas da linguagem de programação utilizada”[4] (BOULAY; O’SHEA; MONK, 1981, p. 265, tradução própria).

Nessa concepção, o estudante não programa em uma linguagem pensando na máquina física, mas sim no modelo mental da *notional machine* inferida pela linguagem utilizada.

Professores muitas vezes utilizam algo próximo de *notional machines* implicitamente ao apresentarem um modelo simplificado do comportamento do

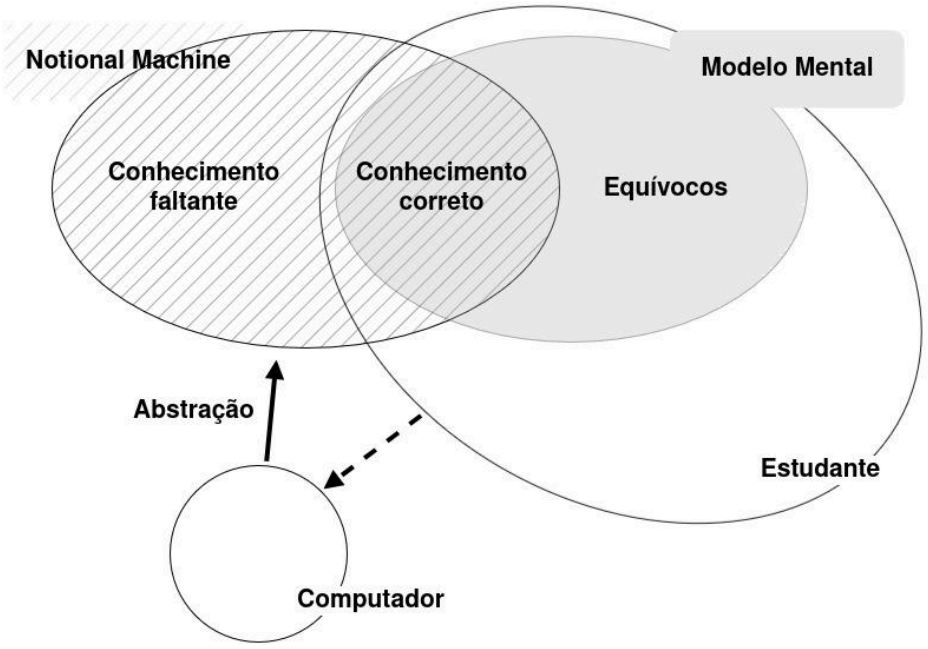
[4] No original: “[...] conceptual computer whose properties are implied by the constructs in the programming language employed”.

computador em suas explicações. O nível de abstração de uma *notional machine* depende então da escolha do professor que pode ser baseada no nível de conhecimento dos estudantes, no propósito ou assunto sendo tratado e na linguagem de programação escolhida.

Seria interessante para uma *notional machine* sendo utilizada para a linguagem de programação C, por exemplo, descrever endereços de memória enquanto que, para uma linguagem orientada a objetos como Java, esse detalhe seria considerado desnecessário.

A Figura 3 mostra a relação entre um estudante, seu modelo mental e uma *notional machine*. Ao escrever um código em uma linguagem de programação em uma máquina física (representado pela linha tracejada na figura), o estudante pensa na execução a partir de seu próprio modelo mental da *notional machine*. Em estudantes iniciantes esse modelo pode ser incompleto, conter equívocos e ser inconsistente. Em contrapartida, a *notional machine* em si é uma abstração consistente e completa do comportamento da máquina no contexto da linguagem e tarefa empregada. Geralmente, quanto mais experiente é o estudante, maior é a interseção de seu modelo mental com a *notional machine* (o conjunto de "conhecimento correto" na Figura 3).

Figura 3 – Relação entre um estudante, seu modelo mental e uma *notional machine*



Fonte: Adaptado de Dickson, Brown e Becker (2020).

Dickson, Brown e Becker (2020) pontuam a importância de separar conceitualmente as *notional machines* de suas visualizações. Uma *notional machine* pode possuir múltiplas visualizações e uma certa visualização pode representar múltiplas *notional machines*. Fazer essa distinção possibilita um uso mais versátil dessas ferramentas por professores, como poder escolher que visualização utilizar depois de já se ter escolhido uma *notional machine*.

De acordo com Sorva (2013), modelos mentais são “executáveis” no sentido em que é possível um estudante acompanhar um sistema em sua mente dado um conjunto de condições iniciais para prever seu comportamento em situações particulares. O problema é que, a partir de “três partes móveis”, acompanhar o estado interno da máquina se torna uma tarefa complicada até para programadores experientes (SORVA, 2013).

Visualizações que representam o estado interno do computador como as contidas em Sistemas de Visualização de Programas (SVPs) permitem acompanhar um número de estados internos bem maior do que simulações mentais. Além disso, a escolha do nível de abstração adequado associado a *notional machine* sendo representada evita que estudantes iniciantes façam má escolhas acerca de quais partes acompanhar e qual nível de abstração escolher, evitando frustrações por conta de carga cognitiva excessiva (SORVA, 2013).

O nível de abstração necessária pode mudar no decorrer de uma única disciplina. Enquanto du Boulay, O’Shea e Monk (1981) enfatizam o quão simples *notional machines* devem ser ao serem introduzidas para estudantes iniciantes, Dickson, Brown e Becker (2020) apontam algo que é possível observar implicitamente em livros didáticos: a gradual adição de detalhes a *notional machine* sendo utilizada, possibilitando trabalhar conceitos mais complexos sem alterar resultados anteriores. Ou seja, é possível encontrar uma relação de inclusão entre *notional machines* em diferentes níveis de abstração.

SVPs podem oferecer diferentes visualizações representando *notional machines* em diferentes níveis de abstração, suprindo as necessidades em múltiplos estágios de uma disciplina ou curso. Essa funcionalidade está presente no SVP Python Tutor discutido na seção 5 e também é discutido na seção 7.1 de trabalho futuro.

3 TRABALHOS RELACIONADOS

Uma série de ferramentas de visualização de software voltados para programadores iniciantes foram desenvolvidas desde os anos 80. Muitas motivadas pela perceptível dificuldade que estudantes têm relacionada às dinâmicas de um programa e conceitos fundamentais de programação (SORVA; KARAVIRTA; MILMA, 2013).

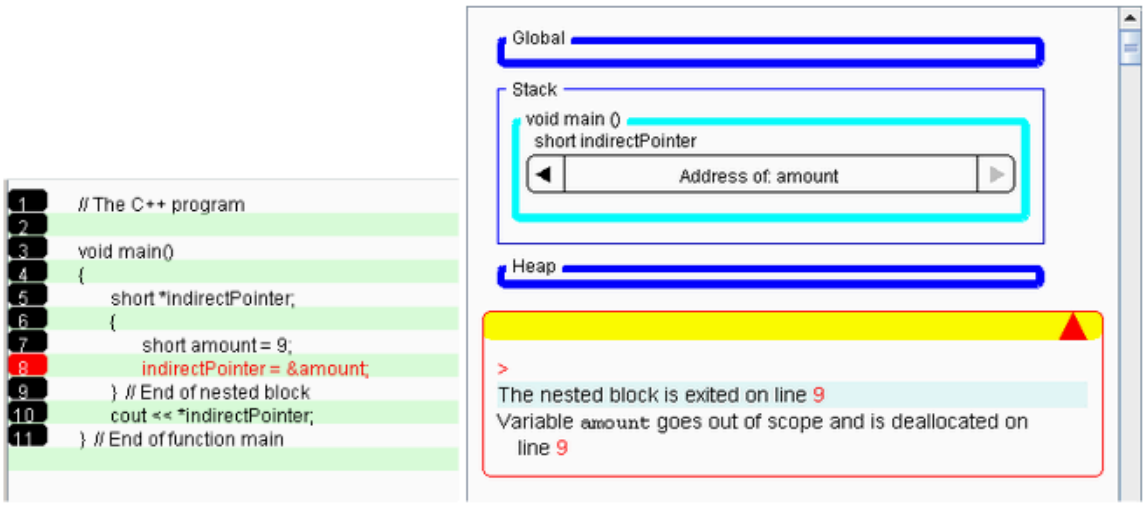
De acordo com Sorva, Karavirta e Milma (2013) uma das maiores dificuldades que estudantes enfrentam é entender a relação entre abstrações de máquinas em execução (*notional machines*) e o código escrito. As visualizações de *notional machines* revelam partes de um programa em tempo de execução que normalmente são invisíveis como referências, ponteiros e recursão.

Todas ferramentas analisadas na revisão de Sorva, Karavirta e Milma (2013) são Sistemas de Visualização de Programas (SVP) genéricos, isto é, ferramentas que focam em visualizar os vários aspectos da execução de programas concretos. SVP especializados, por outro lado, focam na execução de programas concretos ao redor de conceitos ou estruturas particulares. Este trabalho propõe um SVP especializado, focado no conceito de ponteiros e gerenciamento de memória no geral.

Os SVPs aqui mostrados foram selecionados a partir da revisão feita por Sorva, Karavirta e Milma (2013) e também a partir de buscas por palavras chaves como “*program visualization*”, “*c pointers*” e “*program semantics visualizer”* na ferramenta Google Acadêmico e em grandes acervos como o CAPES e SBC-OpenLib da Sociedade Brasileira de Computação (SBC).

Um SVP especializado centrado em ponteiros pode ser visto no artigo de Kumar (2009) no qual ele descreve um sistema *web* que foca na resolução de problemas envolvendo ponteiros utilizando a linguagem de programação C++. O sistema possui um conjunto definido de 8 problemas e possui reconhecimento de diversos erros de programação envolvendo ponteiros como desreferenciamento de ponteiros não inicializados ou que referenciam espaços de memória já desalocados (Figura 4).

Figura 4 – Sistema de Kumar



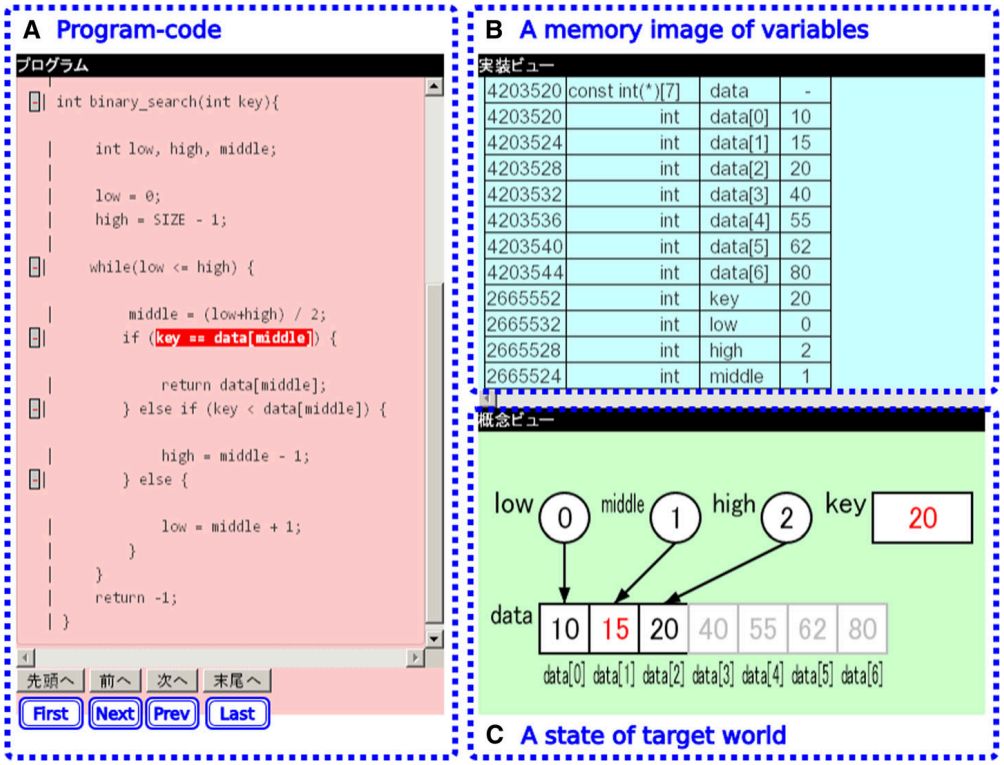
Fonte: Kumar (2009)[5].

O sistema de Kumar (2009) possui uma importante característica que seria a visualização dos diferentes espaços de memória: global, pilha e *heap*. Porém, uma de suas limitações é a rigidez de entradas voltadas para seu conjunto de problemas.

Já Yamashita et al. (2017) desenvolveram um SVP especializado também voltado para ponteiros, chamado TEDViT, que aceita como entrada qualquer programa. Seu sistema possui 3 visões: uma em que consta o código em si, uma que detalha o estado atual da memória e outra focada em visualizar estruturas de dados (Figura 5). Esta última, chamada de "*target world*" é personalizável de acordo com a intenção do docente. Por exemplo: em uma aula sobre a estrutura de dados *array* é preferível que a estrutura seja exibida na horizontal; em uma aula sobre a estrutura de dados pilha é preferível que ela seja exibida na vertical.

[5] A fonte consultada, nessa figura, não é paginada.

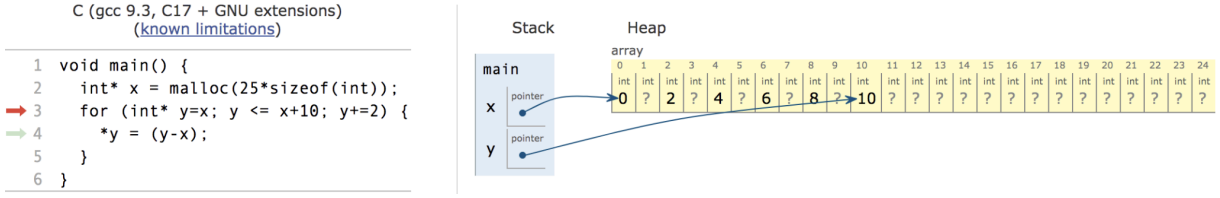
Figura 5 – TEDViT



Fonte: Yamashita et al. (2017, p. 4).

Mais recentemente, GUO (2021) relata em seu artigo o amplo uso do seu SVP genérico desenvolvido em 2013: Python Tutor. Apesar do nome, a ferramenta possui suporte para diversas linguagens de programação, em especial C e C++. A visualização de ponteiros e de espaços de memória nessas linguagens é equivalente aos sistemas mostrados anteriormente (Figura 6).

Figura 6 – Python Tutor



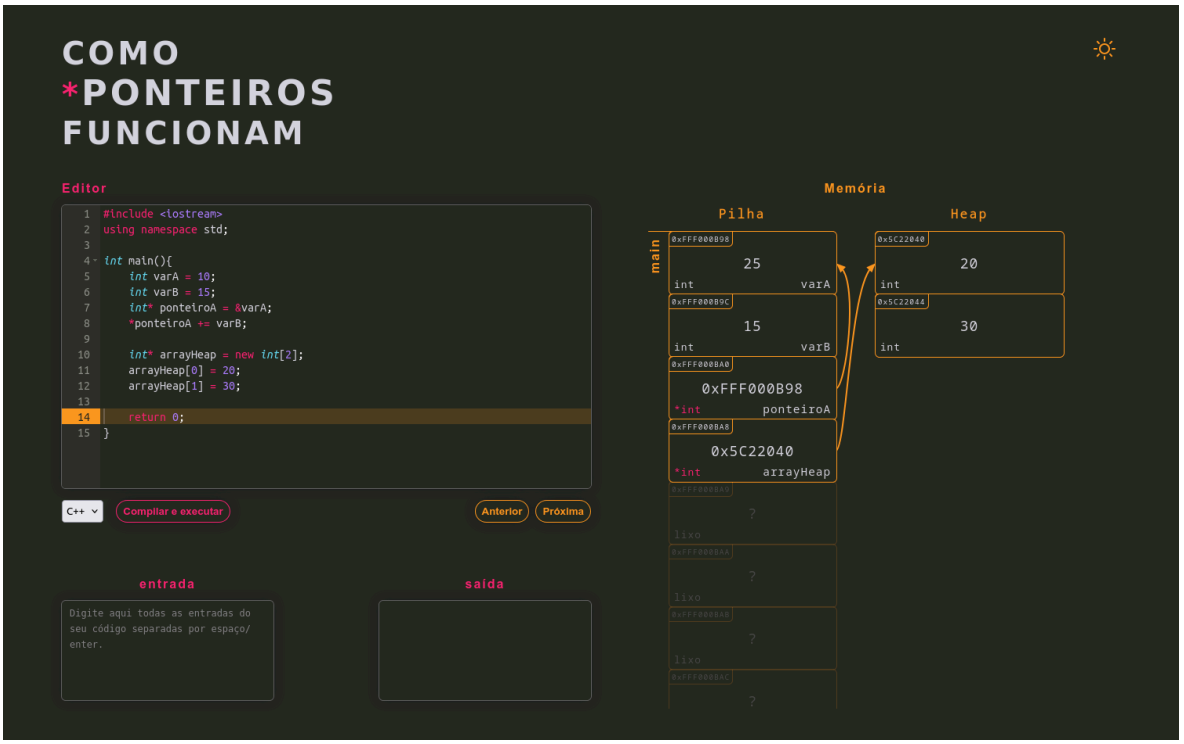
Fonte: Guo (2021)[6].

Como é descrito na seção seguinte, a solução proposta por este trabalho utiliza uma versão do Valgrind adaptada por Guo (2021) e utilizada no Python Tutor.

[6] A fonte consultada, nessa figura, não é paginada.

O sistema proposto por este trabalho (Figura 7) diverge das soluções apresentadas acima ao apresentar a memória do computador de maneira contínua e na forma de caixas, dessa forma se aproximando das visualizações da memória muitas vezes desenhadas em salas de aula por professores. Sua forma generalizada também se permite adaptar a diferentes *notional machines* utilizadas por diferentes professores em diferentes contextos e níveis de abstração.

Figura 7 – Sistema proposto

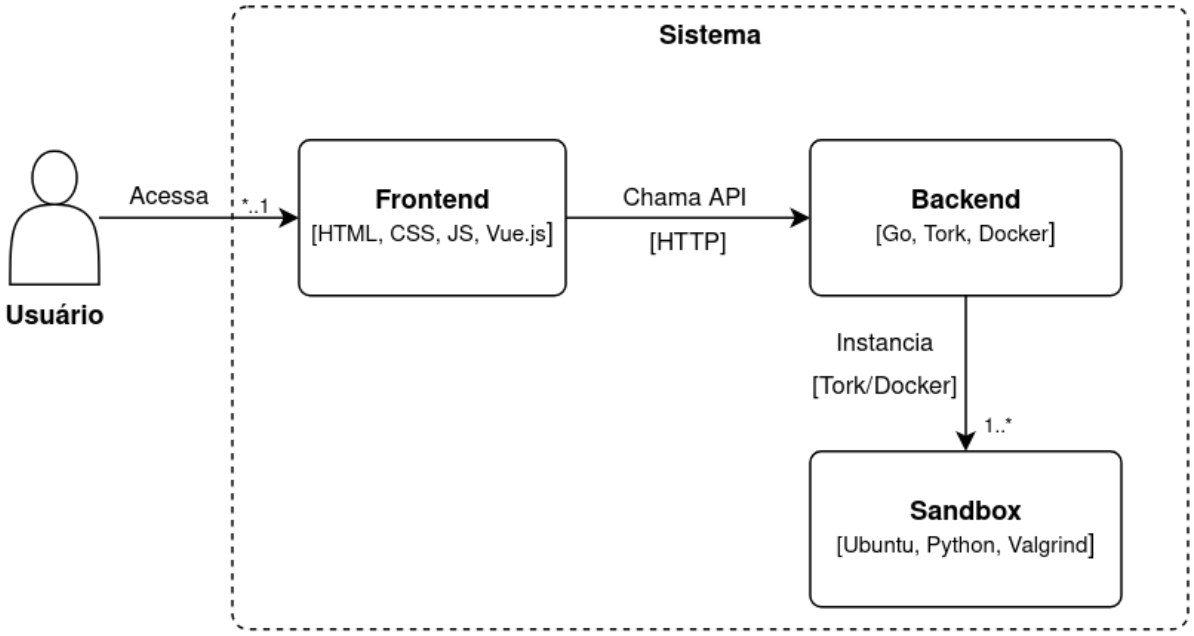


Fonte: Elaborado pelo autor (2025).

4 SOLUÇÃO PROPOSTA

O sistema que este trabalho propõe é um serviço para ajudar a visualizar a execução de um programa na memória com foco no conceito de ponteiros. Ele é dividido em duas partes: um *frontend*, o documento HTML em que o usuário final tem contato através de um navegador *web* e um *backend* que executa os códigos do usuário em um ambiente seguro, chamado de *sandbox*. A arquitetura geral do sistema com suas tecnologias pode ser observada na Figura 8.

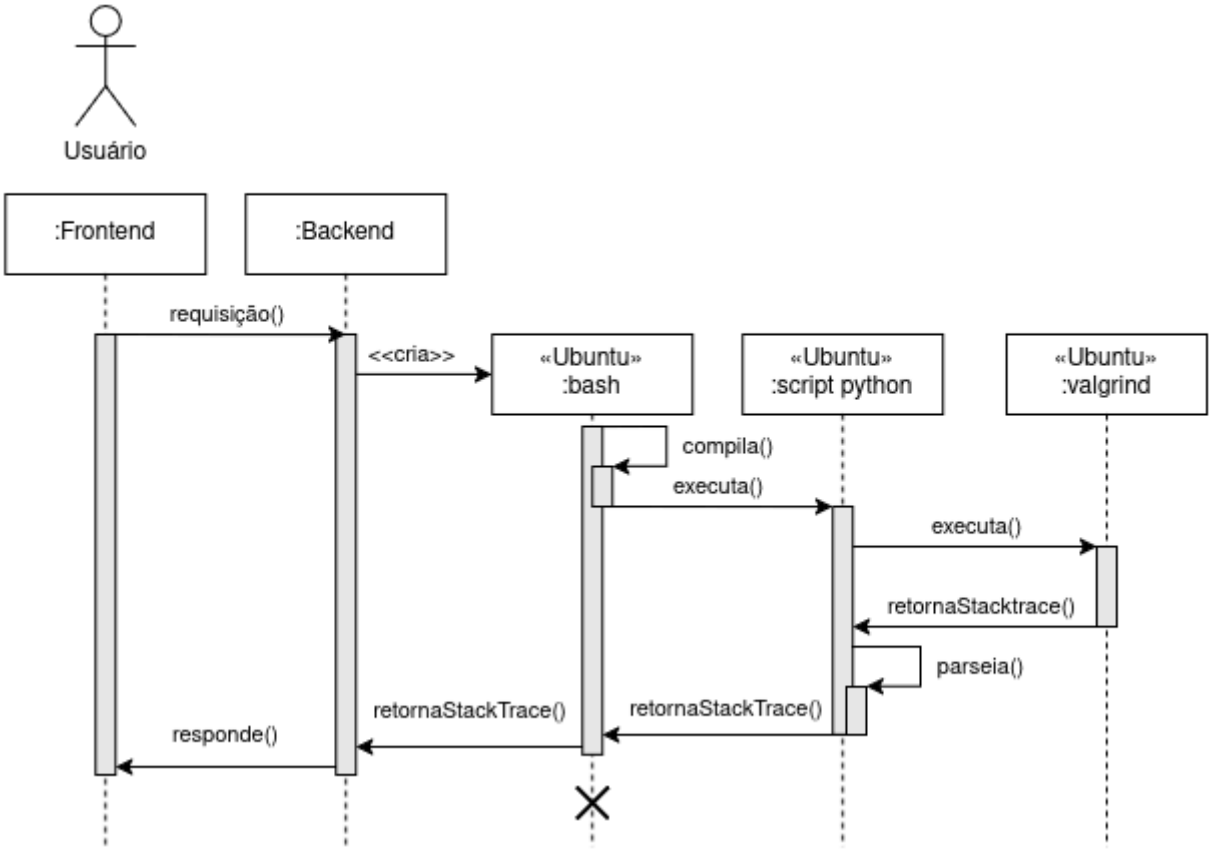
Figura 8 – Diagrama da arquitetura



Fonte: Elaborado pelo autor (2025).

O *frontend* e o *backend* se comunicam através de requisições HTTP. O primeiro envia um código escrito pelo usuário para o *backend* que pode conter ou não entradas do programa já preparadas. O *backend*, por sua vez, compila o código e o executa através de uma versão alterada do depurador de código Valgrind. O depurador, então, gera um relatório contendo o estado de cada variável em cada passo de execução do programa, chamado de *stacktrace*. O *stacktrace* é finalmente enviado de volta para o *frontend*, para poder gerar uma visualização da memória. Este fluxo de dados pode ser observado no diagrama de sequência da Figura 9.

Figura 9 – Diagrama de sequência do sistema



Fonte: Elaborado pelo autor (2025).

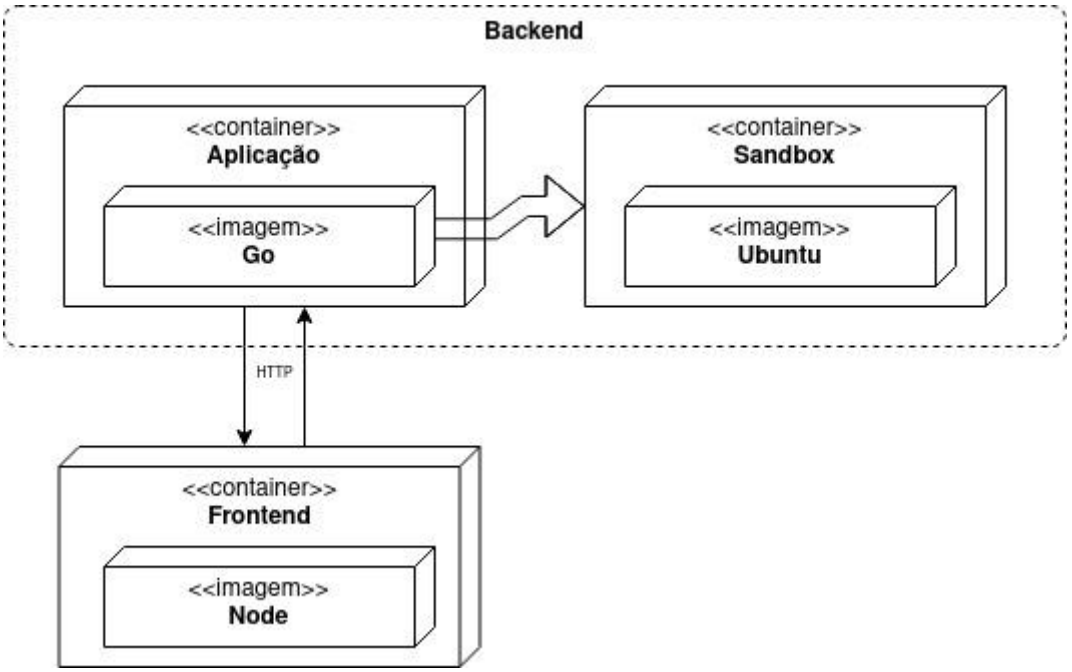
A primeira seção discute as tecnologias utilizadas nesse sistema enquanto a segunda se aprofunda no funcionamento e usabilidade do *website*. A seção final decorre das dificuldades e soluções pensadas no que tange a acessibilidade do sistema para pessoas com deficiências.

4.1 Tecnologias

O sistema é baseado em contêineres Docker. Os contêineres são processos que servem como ambientes de execução de programas isolados da máquina que os hospeda e de outros contêineres. Por simularem uma máquina a nível de sistema operacional, eles consomem menos recursos e iniciam de forma mais rápida que máquinas virtuais, que utilizam virtualização de *hardware* (RAD, 2017).

Contêineres Docker são instanciados a partir de arquivos chamados imagens. A Figura 10 ilustra os *containers* que constituem o *backend* e o *frontend* do sistema com suas respectivas imagens.

Figura 10 – Diagrama de *deployment*



Fonte: Elaborado pelo autor (2025).

4.1.1 Frontend

O primeiro protótipo do sistema envolvia somente uma página HTML estática com uma visualização da memória feita na biblioteca gráfica p5.js[7]. O editor de texto era um elemento *textarea* e seu conteúdo era processado utilizando expressões regulares do JavaScript. Esse interpretador de C na época conseguia identificar declarações e uso de variáveis, seus tipos e avaliar expressões aritméticas simples.

Com a necessidade de representar *frames* de chamadas de funções e seus escopos, foi analisado que a complexidade de se manter um interpretador era muita alta e outras alternativas foram buscadas. Ao fim, a arquitetura atual com a compilação do código do usuário e sua subsequente execução pelo Valgrind em um servidor separado foi escolhida.

Além disso, como a biblioteca p5.js gerava uma visualização baseada em imagens no formato SVG, ela foi substituída pelo *framework* Vue. As imagens em SVG dificultavam a interação com os elementos e os tornavam completamente

[7] Disponível em: <https://p5js.org> Acesso em: 24 jun, 2025

inacessíveis a leitores de tela. Em contraste, os componentes reutilizáveis do Vue geram elementos HTML que, quando utilizados de maneira apropriada, facilitam a interação e a acessibilidade.

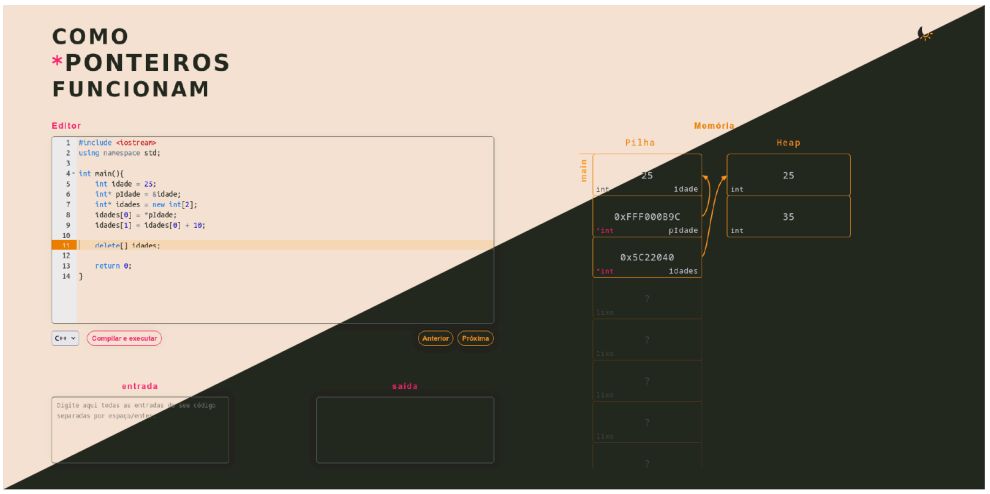
Vue é utilizado no formato de *script* isolado local, permitindo sua coexistência com as partes estáticas da aplicação sem adicionar complexidade como uma etapa de transpilação, por exemplo.

O editor de código presente no *site* também vem de uma ferramenta externa: Ace[8]. Dentre suas funcionalidades, estão presentes: coloração das sintaxes de C e C++, dobragem de código, autocomplemento, uso de *snippets* e aplicação de temas. Todas elas são características que pessoas desenvolvedoras esperam de qualquer editor de código e, portanto, tornam o site familiar para esse público.

Além dos pontos citados, o editor de código Ace segue padrões que o tornam acessível para pessoas usuárias de leitores de tela ou que usam o teclado como única forma de navegação. Situações como “armadilhas de foco” (*focus traps,* em inglês) são evitadas com a possibilidade de se utilizar a tecla ESC para sair do editor (e da anunciação dessa possibilidade pelo leitor de tela).

Outra característica que o *frontend* possui é a opção de mudar o seu esquema de cores. O tema padrão é “escuro” mas é possível mudar para o tema “claro” clicando no ícone de sol no canto superior direito. O tema claro facilita a leitura para algumas pessoas e tem se afirmado como melhor opção ao exibir o *site* em um projetor em sala de aula (Figura 11).

Figura 11 – Diferença dos temas escuro e claro



Fonte: Elaborado pelo autor (2025).

[8] Disponível em: <https://ace.c9.io> Acesso em: 24 jun, 2025

O repositório do código do *frontend* está disponível na plataforma Github[9].

4.1.2 Backend

O *backend* é feito em volta da ferramenta Tork[10], um gerenciador de fluxo de trabalho distribuído. Ele permite executar programas com isolamento e limitação de recursos através de contêineres Docker, característica essencial para garantir a segurança ao executar códigos não confiáveis. Por permitir um modo de execução distribuída, também é uma escolha ideal do ponto de vista da escalabilidade.

Como o Tork é feito na linguagem de programação Go, o *backend* foi escrito na mesma linguagem, permitindo facilmente a extensão da ferramenta para os casos de uso necessários do sistema.

Dentre as configurações do Tork, há opções de segurança que restringem a origem das requisições HTTP e o tipo de método aceito. Portanto, quaisquer requisições que não provenham do IP do frontend ou que não utilizem o método HTTP POST não são aceitas pelo *backend*.

Outro estratégia de segurança foi a higienização da entrada de dados a ser enviada para o programa escrito pelo usuário, permitindo somente caracteres latinos ou números separados por espaços. Essa verificação é feita com a biblioteca de expressões regulares presente na linguagem Go.

Ao receber o código do usuário via uma requisição HTTP, o *backend* inicia uma tarefa no Tork utilizando um contêiner gerado pela imagem do sistema operacional Ubuntu. Esse contêiner é gerenciado pela mesmo Docker Engine que instancia o contêiner da aplicação em Go. Tal prática é empregada com o intuito de evitar o *docker in docker*, isto é, a execução de um contêiner Docker dentro de outro contêiner, prática não recomendada pela própria equipe de desenvolvimento do Docker (PETAZZONI, 2020).

O Tork permite limitação de memória, tempo de execução e uso de CPUs que uma tarefa tem à disposição. Para o protótipo desenvolvido foi estabelecido um tempo de execução máxima de 20 segundos, uso de 1000 MB de memória RAM e 1 núcleo de CPU.

---

[9] Disponível em: <https://github.com/arturo32/HowPointersWork> Acesso em: 24 jul, 2025
[10] Disponível em: <https://www.tork.run> Acesso em: 24 jun, 2025

Além dessas limitações de recursos, uma limitação de 300 caracteres foi definida para códigos enviados ao *backend*. Esse tamanho foi considerado apropriado para trechos de código com fins educativos e também acaba reduzindo o uso de recursos pelo compilador e o Valgrind.

Dentro da tarefa do Tork, o código é compilado utilizando o compilador gcc ou g++*,* dependendo da linguagem escolhida. Se algum erro de compilação ocorrer, a tarefa é finalizada imediatamente e as informações de linha e coluna na mensagem de erro são extraídas e enviadas juntas de volta ao *frontend*.

Caso a compilação seja bem-sucedida, um *script* escrito em Python executa o programa compilado em uma versão customizada da ferramenta de depuração Valgrind. Essa versão do Valgrind foi desenvolvida por Philip Guo para ser utilizada no Python Tutor (GUO, 2013), um SVP discutido na seção 4.

Ao finalizar a execução, um relatório contendo detalhes de todos os passos de execução é gerado. O mesmo *script* em Python formata esse relatório em algo mais organizado. A tarefa finaliza e o relatório é enviado de volta para o *frontend*.

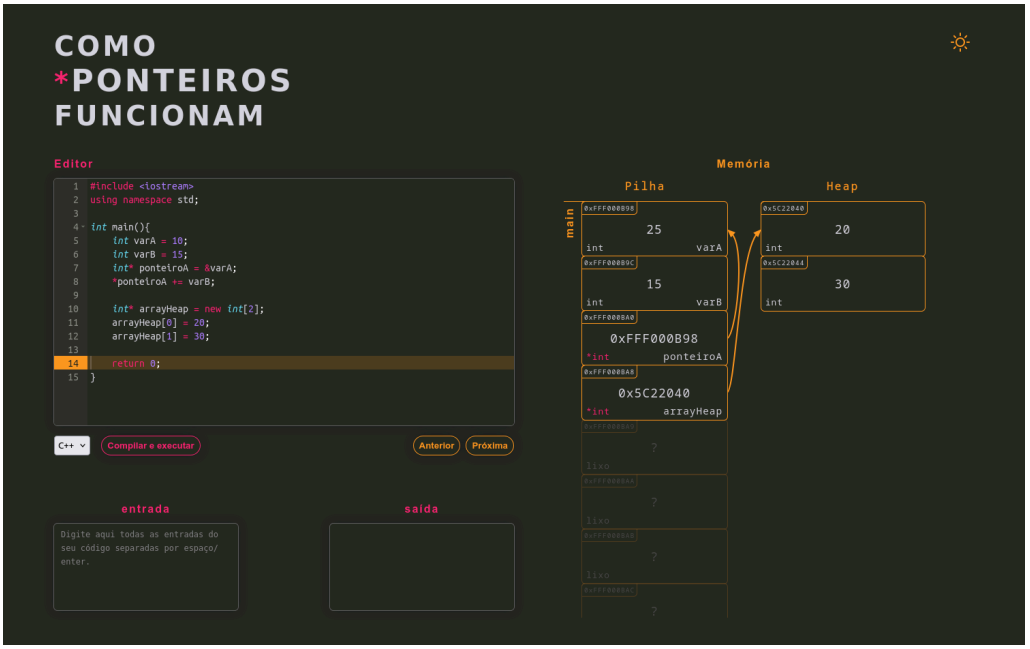
Assim como o *frontend*, o código do *backend* está disponível na plataforma Github[11].

4.2 Funcionamento

O sistema que este trabalho propõe, como dito na seção anterior, é um site composto de duas visões que compartilham a mesma tela: um editor de código que suporta as linguagens C e C++ e uma representação da memória da *notional machine* conectada ao código escrito no editor. A Figura 12 apresenta essas visões na esquerda e direita, respectivamente.

[11]Disponível em: <https://github.com/arturo32/HowPointersWork-server> Acesso em: 24 jun, 2025

Figura 12 – Protótipo da solução



Fonte: Elaborado pelo autor (2025).

Quando um usuário digita seu código no editor de código e pressiona o botão “compilar e rodar” o site entra no modo de execução. Nesse modo aparecem dois botões abaixo do editor de texto: “anterior” e “próxima”. Eles controlam o estado atual do programa, alterando a linha a ser executada. Se uma linha altera o estado da memória, essa alteração é instantaneamente representada na visualização da memória à direita.

A memória é dividida em duas seções: pilha e *heap*. A pilha utiliza a parte estática da memória relacionada ao programa: um espaço reservado ao programa com um tamanho conhecido em tempo de compilação. As variáveis locais e o gerenciamento de chamadas de funções são armazenados nela. Já a *heap* representa a memória dinâmica do programa, requisitada ao sistema operacional em tempo de execução.

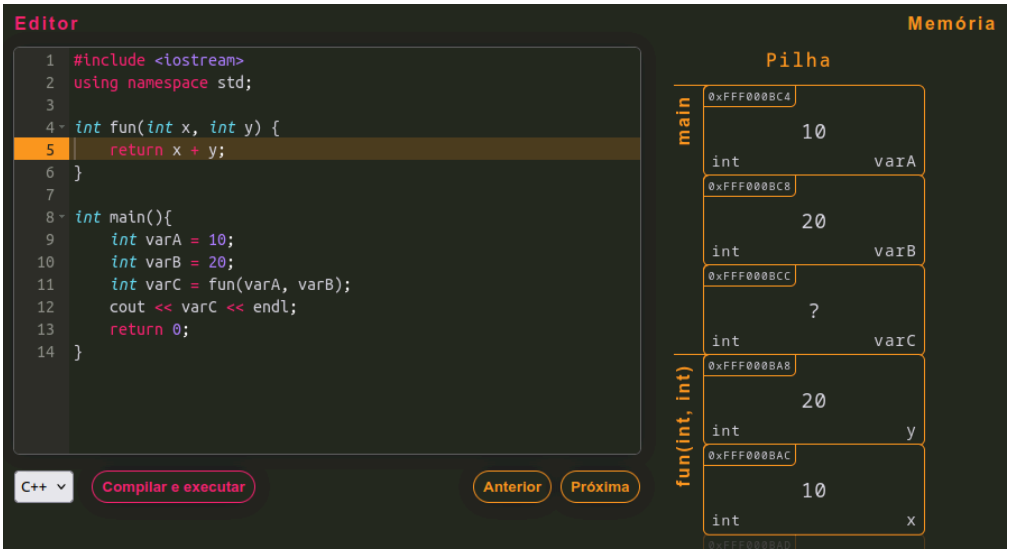
Cada seção da memória é dividida em células de memória. Cada célula possui um endereço (parte superior esquerda), um tipo (inferior esquerda) e um nome (inferior direita). Esses detalhes podem ser vistos na Figura 13.

Figura 13 – Célula de memória

Fonte: Elaborado pelo autor (2025).

Quando uma função é chamada, se cria um novo *frame* de função na pilha onde seus parâmetros e variáveis locais são armazenados. Na Figura 14 um código em C++ com uma função chamada “fun” está rodando. Essa função recebe duas variáveis do tipo inteiro e retorna sua soma. Na função principal, “main”, três variáveis são declaradas: “varA”, “varB” e “varC”. Ao executar a linha 11, o fluxo de execução é alterado para dentro da função “fun”, criando-se um novo *frame* de função na pilha.

Figura 14 – *Frames* de função



Fonte: Elaborado pelo autor (2025).

Nesse mesmo exemplo é possível observar que variáveis não inicializadas, como o caso de “varC” antes da função “fun” retornar, possuem o caractere “?” como valor, indicando que seu conteúdo é desconhecido e sem significado.

Quando se declara um ponteiro, sua representação na visualização aparece de forma similar a uma variável simples, tendo como valor o endereço da variável

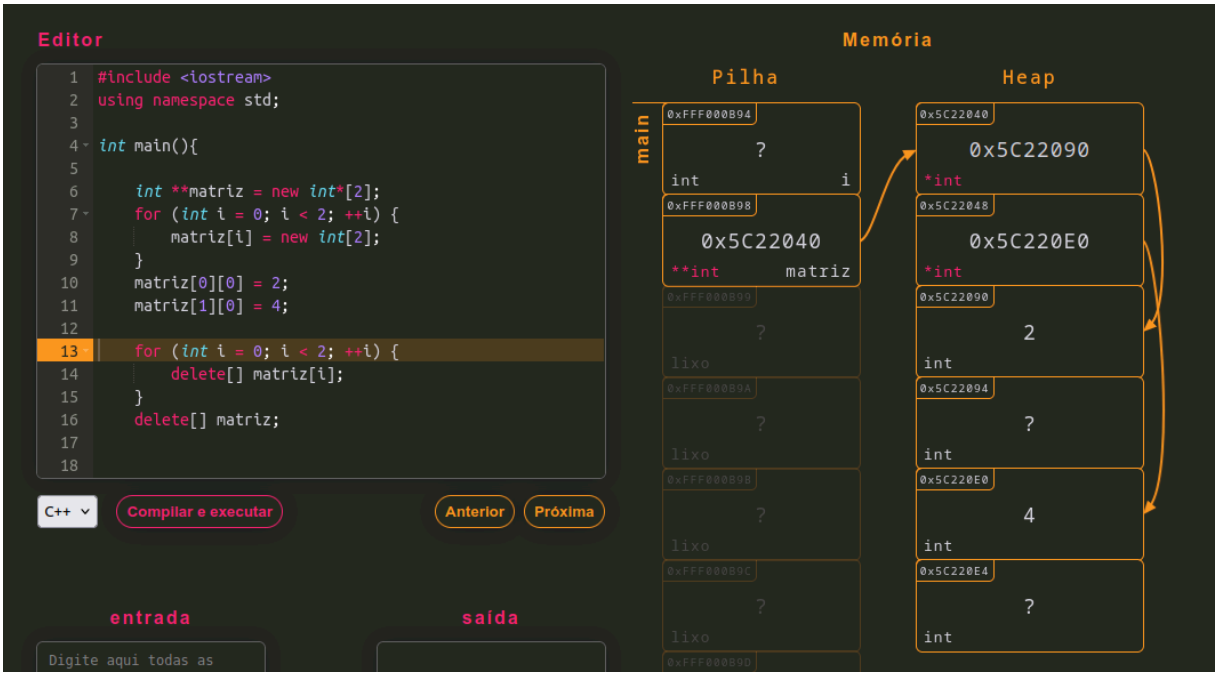
que está apontando e como seu tipo destacado com a cor rosa. Uma seta representa a conexão do ponteiro com o espaço de memória que ele aponta (Figura 15).

Figura 15 – Uso de ponteiros simples

Fonte: Elaborado pelo autor (2025).

A Figura 16 mostra como a *heap* e matrizes são representadas em um código com alocação dinâmica.

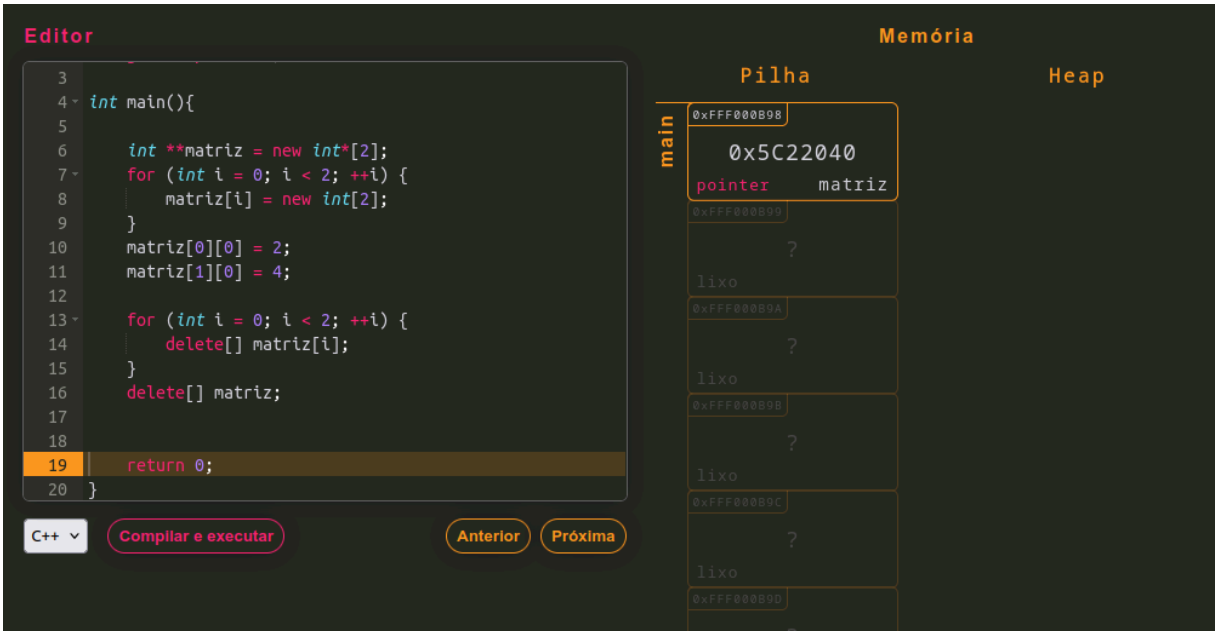
Figura 16 – Uso de ponteiros com alocação dinâmica e matrizes



Fonte: Elaborado pelo autor (2025).

Continuando com a execução do código da Figura 15, a Figura 16 mostra que, ao desalocar um espaço de memória apontado por um ponteiro, esse ponteiro mantém o seu valor, mas é possível notar que não existe mais nenhum espaço de memória associado a ele e a tentativa de acesso a esse espaço não existente[12] resultaria em erro. Esse estado do ponteiro é conhecido como *dangling pointer* (Figura 17).

Figura 17 – Desalocação de memória e *dangling pointers*

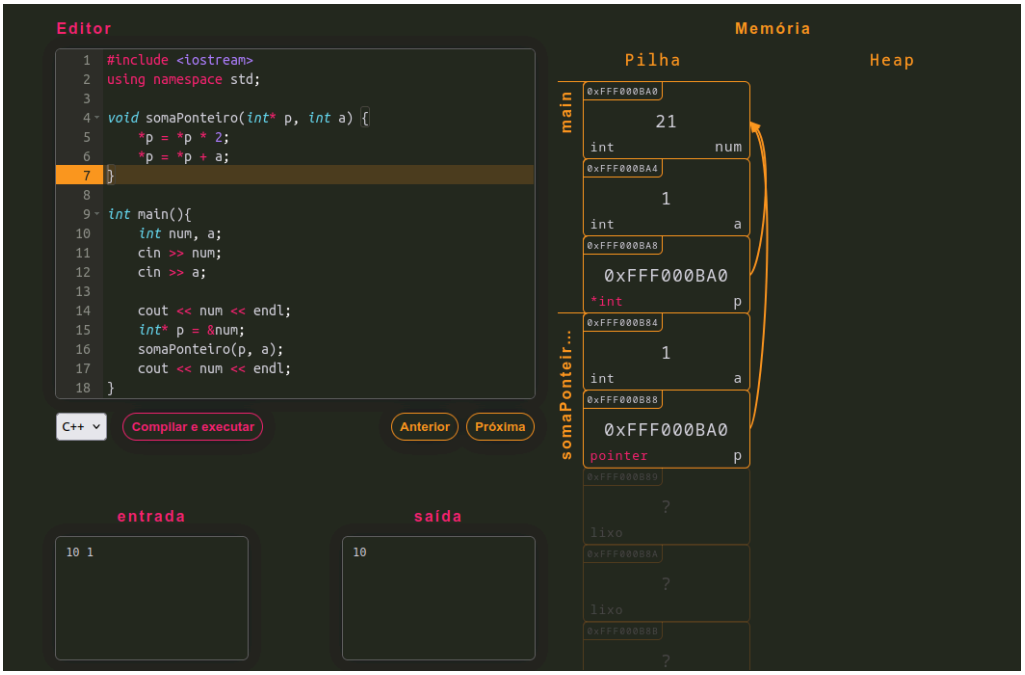


Fonte: Elaborado pelo autor (2025).

A Figura 18, por sua vez, mostra como a entrada e saída de dados funciona. A entrada deve ser preenchida antes da execução, com diferentes valores separados por espaço ou quebra de linha. A saída funciona durante a execução, como é possível notar ao ver o resultado da linha 14 no bloco da saída.

[12] O espaço, na verdade, ainda existe, mas o seu acesso pelo programa já não é mais permitido pelo sistema operacional.

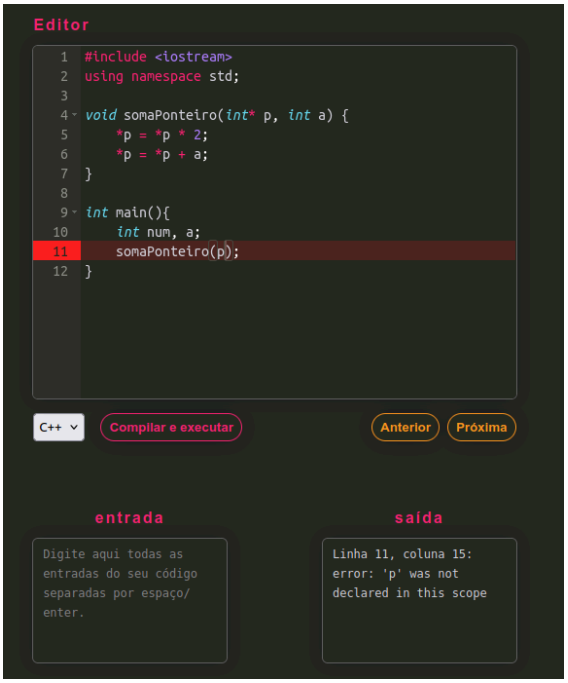
Figura 18 – Entrada e saída de dados do programa



Fonte: Elaborado pelo autor (2025).

Por fim, a Figura 19 mostra como erros de compilação e execução aparecem: destacado de vermelho no editor e com a mensagem de erro sendo exibida no bloco da saída do programa.

Figura 19 – Erro de compilação



Fonte: Elaborado pelo autor (2025).

5 METODOLOGIA

Foram realizados usos supervisionados da ferramenta proposta por este trabalho em salas de aula na Universidade Federal do Rio Grande do Norte (UFRN), no âmbito da disciplina de Introdução a Técnicas de Programação, oferecida a ingressantes do curso de Ciência da Computação. A turma 2025.1, composta por 37 estudantes, foi ministrada pelo professor André Maurício Cunha Campos e é o objeto de interesse das intervenções e avaliações que serão descritas nesta seção.

As experimentações foram realizadas em um contexto de pesquisa exploratória, cujos principais objetivos foram testar a ferramenta em sala de aula tanto pelo professor como pelos estudantes, coletar relatos de uso e analisar as impressões de estudantes acerca da ferramenta e seus modelos mentais após as intervenções. O conhecimento dos estudantes acerca do conceito de ponteiros no decorrer das aulas também foi avaliado.

Durante o uso da ferramenta por estudantes, o uso da memória do servidor da ferramenta foi coletado para fins de análise de sua eficiência.

5.1 Uso geral e impressões

O período entre 15/04/2025 e 22/05/2025 foi destinado ao uso da ferramenta em aulas expositivas de assuntos diversos de programação como funções, escopos de variáveis e tipos de passagens de parâmetros.

O uso da ferramenta foi incentivado fora das aulas, com o detalhe que ela só poderia ser utilizada dentro da universidade por conta das restrições de rede do servidor, que estava hospedado em um serviço de nuvem da própria universidade, o IMDCloud.

O período entre 24/06/2025 e 26/06/2025 foi destinado ao uso da ferramenta tanto em aulas expositivas como no uso pessoal por estudantes com foco no conceito de ponteiros. Os usos da ferramenta nesse período foram realizados em duas aulas.

5.2 Uso com ponteiros pelo professor

A primeira, realizada no dia 24/06/2025, envolveu uma aula expositiva apresentando conceitos de: definição de ponteiros, associação com *arrays*, passagem de referências em funções e aritmética de ponteiros. Na mesma aula, uma avaliação, feita por meio do Google Forms com quatro perguntas que questionavam a saída de pequenos programas, foi aplicada na turma. As questões deste formulário estão disponíveis no Apêndice A. Cada questão tinha um objetivo específico, como mostrado na Tabela 1.

Tabela 1 - Objetivos das questões do primeiro formulário

Questão	Objetivo
1	Avaliar a compreensão de acesso indireto quando usamos ponteiros
2	Avaliar o nível de confusão ou desambiguidade gerada pelas várias semânticas do operador “*”
3	Avaliar o entendimento do acesso aos índices de um array usando "saltos" (endereço inicial + salto)
4	Avaliar o entendimento sobre a aritmética de ponteiros em um array

Fonte: Elaborado pelo autor (2025).

A atividade foi realizada sem acesso a ferramenta ou editores de código. Os estudantes tinham que percorrer (traçar) o código mentalmente. Os resultados desse teste e deduções dos raciocínios utilizados para respondê-lo são apresentados na seção de resultados.

5.3 Uso com ponteiros pelos estudantes

A segunda aula, realizada no dia 26/06/2025, envolveu uma aula expositiva acerca dos conceitos de alocação dinâmica. Foi seguida por uma avaliação com o uso, pela primeira vez, da ferramenta de forma direta pelos estudantes.

A avaliação aplicada foi dividida em três partes: a primeira, envolvia três questões discursivas que pediam o desenvolvimento de programas que utilizavam alocação dinâmica de *arrays*. A segunda e terceira etapas tinham o intuito de recolher impressões sobre o uso da ferramenta na resolução das questões da primeira parte (uso individual) e no uso da ferramenta nas aulas anteriores (pelo professor).

A segunda etapa continha quatro afirmações onde era pedido para o estudante responder seu grau de concordância com elas a partir de uma escala

Likert de cinco opções: “Discordo totalmente”, “Discordo parcialmente”, “Neutro”, “Concordo parcialmente” e “Concordo totalmente”. Uma questão discursiva pedia que, caso a pessoa tivesse discordado em algum grau da quarta afirmação “Usaria a ferramenta novamente”, que explicasse o motivo. A última questão, também discursiva, pedia críticas ou sugestões para o uso individual da ferramenta.

A terceira etapa continha 3 afirmações na mesma configuração que a segunda etapa, só que em relação ao uso da ferramenta pelo professor. A última questão, discursiva, também pedia críticas e sugestões para esse uso.

Esse formulário se encontra na íntegra no Apêndice B e seus resultados são apresentados na seção seguinte.

6 RESULTADOS

Os resultados aqui discutidos se referem ao uso da ferramenta por estudantes e o professor citados na metodologia e realizados entre as datas 15/04/2025 e 26/06/2025.

6.1 Uso geral

O primeiro encontro com a ferramenta gerou dúvidas imediatas acerca de ponteiros por parte dos estudantes, visto que o termo está presente no título do *website*. Sugestões foram feitas por estudantes no sentido de se adicionar textos explicativos espalhados pelo *site*. Essas sugestões são discutidas na seção de trabalhos futuros na conclusão deste trabalho.

Não foi registrado nenhum uso da ferramenta fora de sala de aula pelo registro de *logs* do servidor, mesmo esse uso sendo incentivado. O autor supõe que isso pode se ter dado por alguns fatores: a falta de disposição de estudantes em traçarem seu próprio código ou de usarem debugadores, como é notado por Sorva (2013) e a falta de interesse geral por estudos fora da aulas. A dificuldade de acesso ao *site*, que só pode ser acessado pela rede da universidade e a falta de um nome DNS associado ao IP, também foi levado em consideração.

6.2 Uso com ponteiros

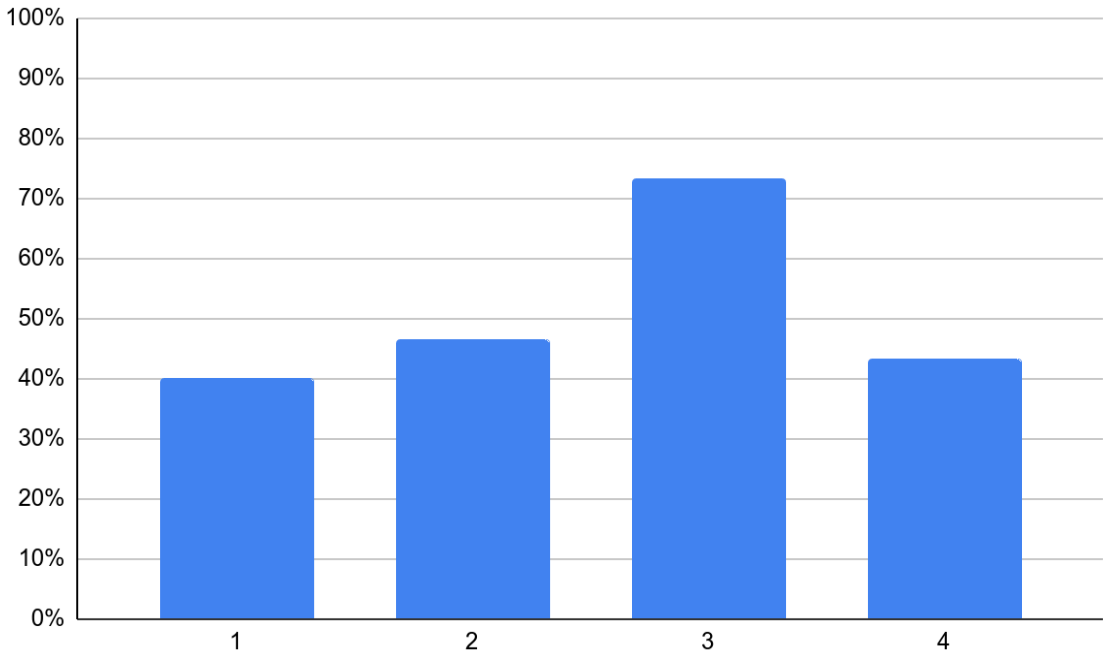
6.2.1 Primeira aula

Durante a primeira aula sobre ponteiros, dúvidas interessantes foram levantadas pelos estudantes. Após explicações feitas pelo professor utilizando slides e a solução proposta por este trabalho, um estudante perguntou se era possível modificar o endereço de memória de uma variável. Essa indagação demonstra uma falta de alinhamento do modelo mental dessa pessoa com a *notional machine* representada pela visualização apresentada.

Uma hipótese para o que pode ter contribuído para esse problema de compreensão específico é a falha da visualização em apresentar componentes imutáveis ou de forma similar a propriedades de variáveis como seu nome ou valor.

Um total de 30 estudantes responderam o formulário de quatro questões (Apêndice A) com uma média de 2,33 questões acertadas por estudante. A primeira questão, que exercitava alterações indiretas de variáveis por meio de ponteiros, foi a que apresentou menos acertos, com 40% (12) dos estudantes acertando ela, seguido pela quarta questão com 13 acertos, a segunda com 14 acertos e a terceira com 22 acertos. O percentual de acerto de cada questão pode ser visualizado no Gráfico 1.

Gráfico 1 - Percentual de acertos por questão do primeiro formulário



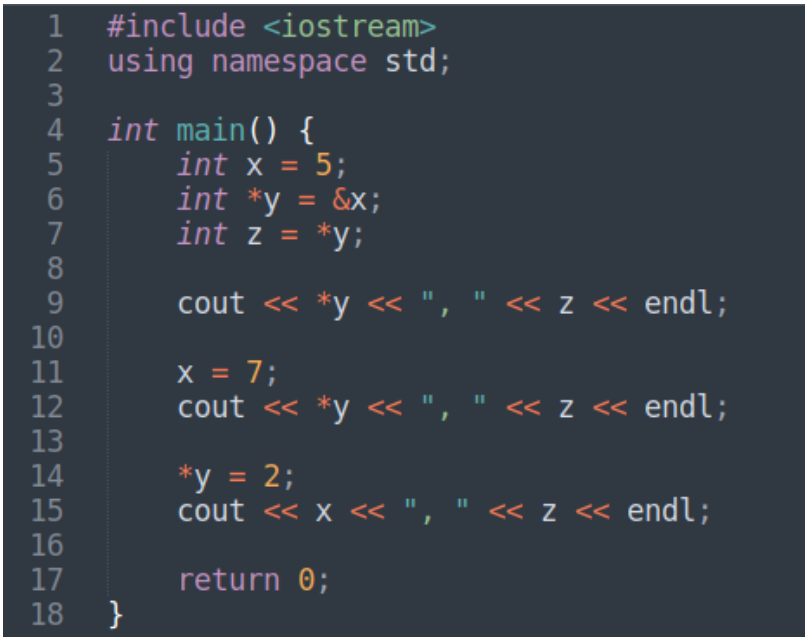
Fonte: Elaborado pelo autor (2025).

Algumas das questões, como a 2, são intencionalmente confusas, para que estudantes tenham contato com a máxima variação sintática possível. Isso, somado ao fato desse ser o primeiro contato com o tópico, pode justificar os resultados com pouca quantidade de acertos.

Uma análise dos erros nas questões 1 e 2 foi feita pelo professor para entender os problemas de compreensão dos estudantes.

A questão 1 (Figura 20) tinha como resposta esperada as *strings* “5, 5”, “7, 5” e “2, 5”, correspondentes às linhas 9, 12 e 15, respectivamente.

Figura 20 – Questão 1



```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int x = 5;
6      int *y = &x;
7      int z = *y;
8
9      cout << *y << ", " << z << endl;
10
11     x = 7;
12     cout << *y << ", " << z << endl;
13
14     *y = 2;
15     cout << x << ", " << z << endl;
16
17     return 0;
18 }
```

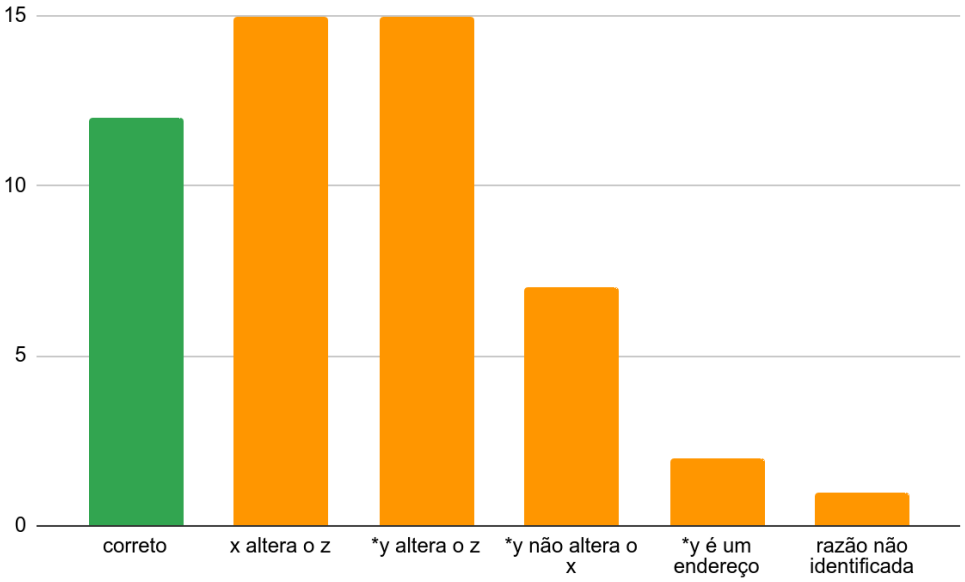
Fonte: Elaborado pelo autor (2025).

Quinze das respostas associaram a *string* “7, 7” a linha 12, levando a pensar que estudantes associaram uma mudança na variável “x” a uma mudança na variável z, mesmo ambas sendo variáveis simples do tipo “int”. O número 2 também foi associado a impressão do valor da variável “z” na linha 15 por 15 pessoas, implicando no entendimento que uma alteração no valor do espaço de memória apontado pelo ponteiro “y” impacta no valor da variável “z”.

Ambos problemas de compreensão podem ser associados ao entendimento da linha 7 como uma associação entre a variável “z” e o ponteiro “y”. Isso, somado às respostas que não associaram a alteração do valor apontado por “y” com uma alteração no valor de “x”, indica uma má compreensão acerca do operador de desreferenciamento.

Esses e outros problemas de compreensão podem ser visualizados no Gráfico 2.

Gráfico 2 - Problemas de compreensão da questão 1



Fonte: Elaborado pelo autor (2025).

A questão 2 (Figura 21) tinha como resposta esperada a *string* “32, 4, 4”, correspondente a linha 11.

Figura 21 – Questão 2

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5      int a = 2;
6      int b = a * a;
7      int *k = &a;
8      int c = *k * *k;
9      *k = b * *k * c;
10
11     cout << a << ", " << b << ", " << c << endl;
12     return 0;
13 }
```

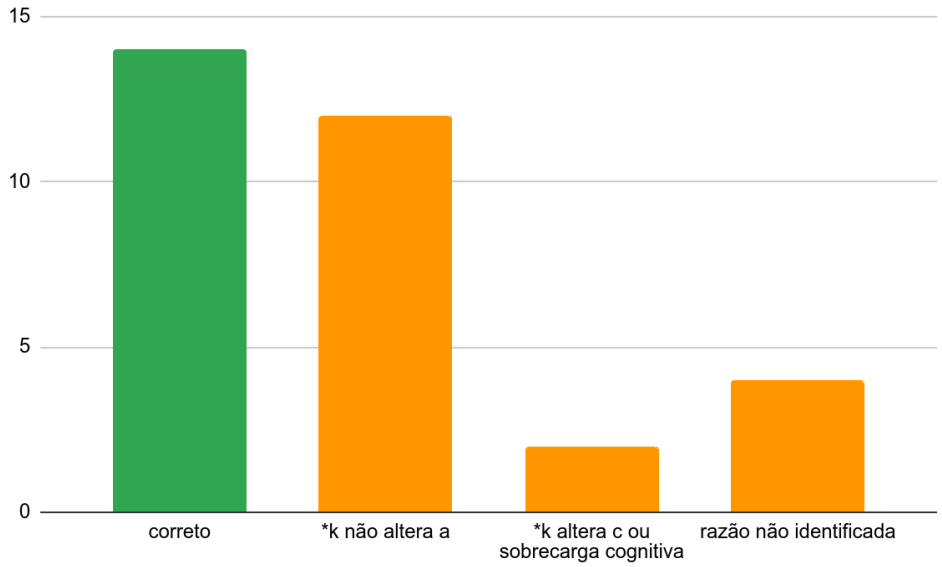
Fonte: Elaborado pelo autor (2025).

Doze respostas continham o número “2” associado à variável “a”, não levando em consideração a alteração do valor de “a” feita através do ponteiro “k” na linha 9. Novamente, o problema parece estar associado a má compreensão do

operador de desreferenciamento. Outras respostas incorretas foram associadas a sobrecarga cognitiva causada pela grande presença de operações.

Os problemas de compreensão do da questão dois podem ser visualizados no Gráfico 3.

Gráfico 3 - Problemas de compreensão da questão 2



Fonte: Elaborado pelo autor (2025).

6.2.2 Segunda aula

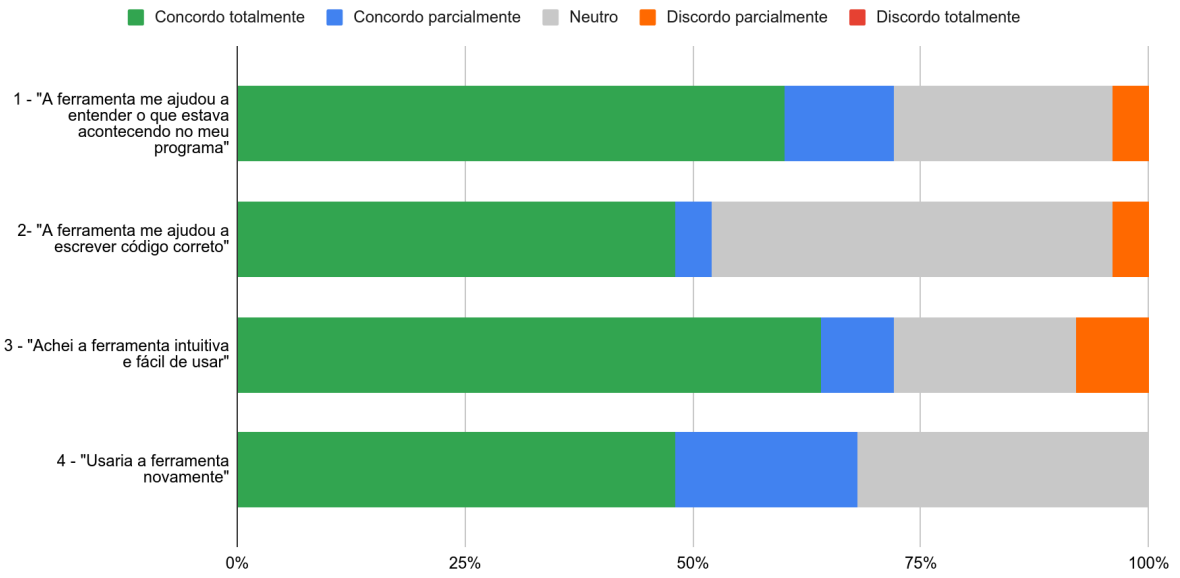
Na segunda aula, sobre alocação dinâmica, foi aplicado o segundo formulário sobre ponteiros (Apêndice B). O intuito dessa atividade era incentivar o uso da ferramenta de visualização de ponteiros para resolver as atividades propostas e depois avaliar seu uso. Mas, após um uso intensivo da ferramenta pela turma por cerca de 7 minutos, o servidor parou de responder as requisições sem nenhum motivo aparente.

Mesmo assim, os estudantes continuaram com o restante do questionário, respondendo a segunda etapa do formulário, referente ao uso pessoal da ferramenta, com base nesse pequeno intervalo de tempo que tiveram contato com ela. No total, 25 estudantes responderam o formulário.

As respostas das questões objetivas da primeira etapa podem ser visualizadas no Gráfico 4. É possível notar que a maioria dos estudantes

concordaram totalmente ou parcialmente com aspectos positivos da ferramenta, enquanto nenhuma pessoa discorda totalmente de qualquer afirmação.

Gráfico 4 - Impressões do uso pessoal da ferramenta



Fonte: Elaborado pelo autor (2025).

Como ninguém respondeu a quarta questão com algum grau de discordância, não houve respostas para a questão que pedia para justificar respostas discordantes da quarta questão.

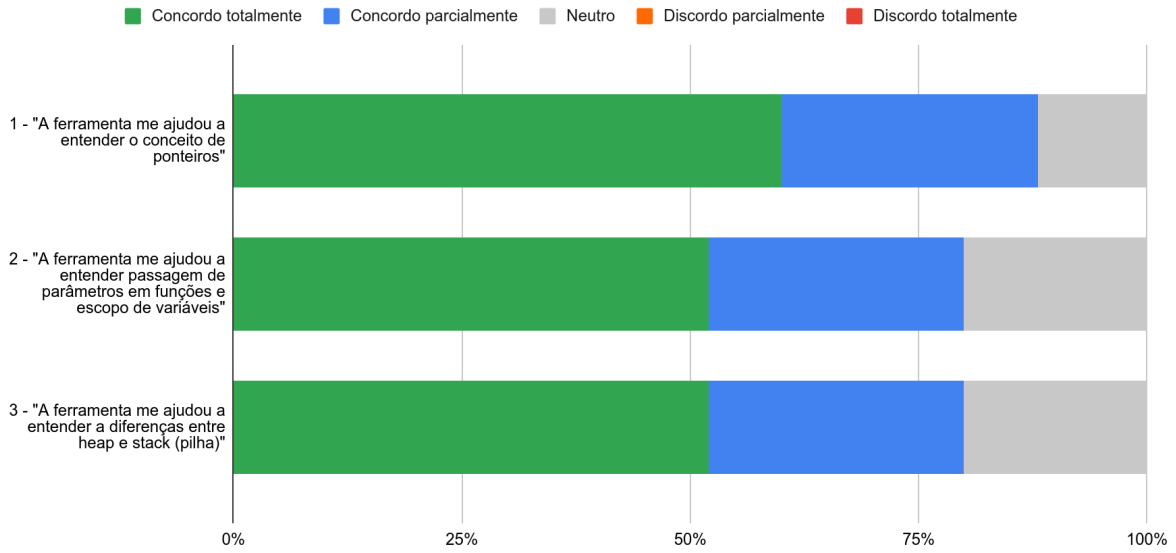
A afirmação que os estudantes menos concordaram foi que “a ferramenta me ajudou a escrever código correto”. Esse resultado pode ser justificado pela falta de analisadores estáticos de código como servidores de linguagem que ajudam apontar erros sintáticos e até semânticos antes da compilação. Isso se alinha com uma das críticas: “[...] achei que o reconhecimento dos erros fica um pouco mais difícil do que no vscode [...] prefiro utilizar o vscode”. Esse fator junto à dificuldade de se adaptar a um novo ambiente de programação possivelmente contribuem também para a baixa concordância total com a afirmação “usaria a ferramenta novamente”.

As principais críticas registradas ao fim desse formulário se referem à “compilação lenta” e a falta de salvamento local do código. A última funcionalidade requerida teria ajudado estudantes a recuperarem seus códigos quando o servidor

da ferramenta parou de responder, deixando-os presos em uma tela de carregamento.

Já a etapa relacionada ao uso da ferramenta pelo professor em aulas expositivas teve respostas mais alinhadas ainda com as afirmações, como é possível observar pelo Gráfico 5. Dessa vez não foi registrada nenhuma resposta em algum grau de discordância.

Gráfico 5 - Impressões do uso da ferramenta pelo professor



Fonte: Elaborado pelo autor (2025).

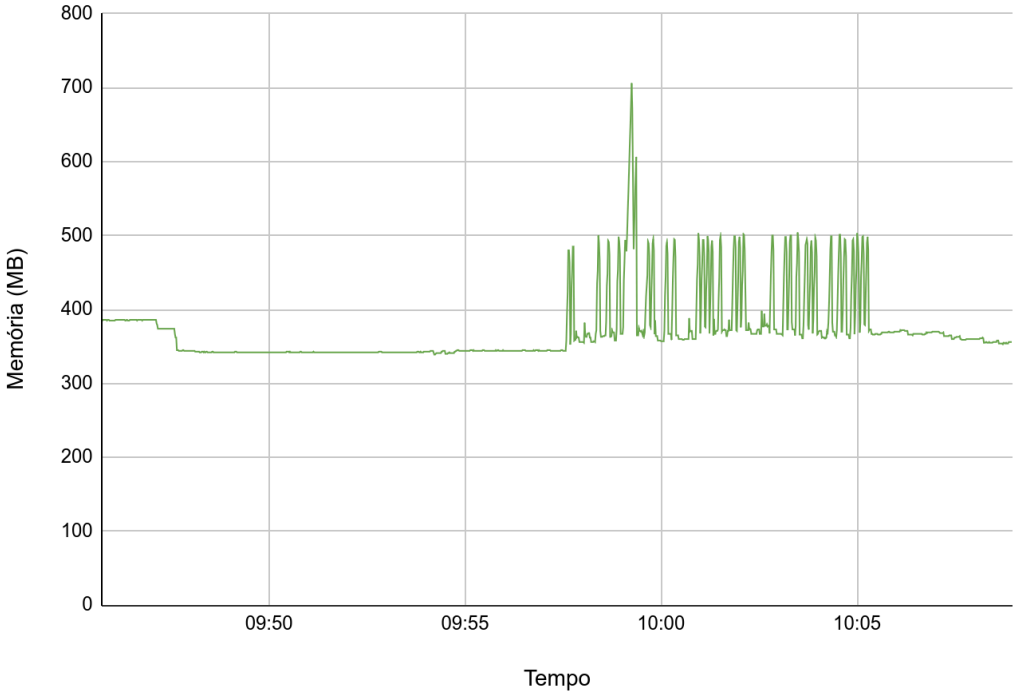
Sugestões foram feitas em relação a setas utilizadas na visualização da memória, indicando que elas poderiam apresentar cores diferentes. Um dos estudantes pontua que achou os conceitos bastante abstratos e que mais estudo e uso desses conceitos em contextos práticos seriam necessários para uma melhor compreensão.

6.2.3 Avaliação do uso da memória

O servidor onde esteve hospedado o *backend* da ferramenta era uma instância de uma máquina virtual do serviço IMDCloud, composta de 2 CPUs virtuais, 3GB de RAM e 20GB de armazenamento. O sistema operacional utilizado foi o Linux Ubuntu 20.04.6 LTS.

O uso da memória foi registrado durante 25 minutos através de um *script bash* que direciona a saída do programa free a cada segundo para um arquivo. O registro foi feito durante o primeiro contato dos estudantes com a ferramenta de forma direta. Durante esse período foram registradas 97 requisições feitas ao servidor, dos quais apenas 65 resultaram na execução do código antes do servidor parar de responder. Esse uso pode ser visualizado no Gráfico 6.

Gráfico 6 - Uso da memória (em MB) no servidor



Fonte: Elaborado pelo autor (2025).

É possível notar que o uso de memória da instância do servidor com o *backend* em repouso é de 350MB, dos quais apenas 48MB são de uso direto pelos containers Docker do *backend*. Ao receber uma requisição, o uso do sistema aumenta para 500 MB em média, uma diferença de 150MB do sistema em repouso.

Um pico de 706MB foi registrado na hora 9:59. O estado de falha do *backend*, então, não pode ser relacionado à falta de memória do sistema visto que em nenhum momento o uso passou mais de um terço da memória disponível.

O uso de memória do frontend, na mesma instância, foi registrado sendo 7,3MB. O uso de memória do sistema ao todo foi considerado satisfatório.

7 DISCUSSÃO

Algo que foi prontamente notado nos usos gerais da ferramenta em sala de aula foi a necessidade de explicar a divisão da memória em “pilha” e “*heap*” pela sua presença visual já no primeiro contato. Uma melhoria que foi pensada ao longo do processo de pesquisa foi a adição de níveis diferentes de abstração à visualização da memória da ferramenta. Dickson, Brown e Becker (2020) apontam que

> Como a notional machine utilizada numa determinada aula só precisa de ser consistente para o material apresentado no momento, pode ser possível começar com uma versão simplificada de um método de visualização que acrescenta gradualmente detalhes à medida que os conceitos se tornam mais complexos num curso ou entre cursos, assegurando que a notional machine específica cumpre sempre o requisito de ser consistente[13] (DICKSON; BROWN; BECKER, 2020, p. 164, tradução própria).

Logo, um nível de abstração em que a memória do computador seja exibida como uma única unidade, sem a presença de endereços de memória, seria útil em aulas iniciais para apresentar conceitos gerais. Outro nível de abstração, que apresenta a divisão da memória e endereços de memória, seria útil para aulas de ponteiros e gerenciamento de memória. Um nível menor ainda, que mostre os conteúdos das células em formato binário, seria útil para evidenciar como diferentes tipos de dados são armazenados pelo computador. Este último nível de abstração também está presente no Python Tutor[14] como uma opção.

Du Boulay (1981) estende o conceito de diferentes níveis de abstrações para as mensagens de erro:

> Mensagens de erro são uma parte crucial do comentário e constituem uma janela importante para a máquina. [...] Assim, se for decidido que um novato não precisa saber do conceito de “stack”, por exemplo, então mensagens de “stack overflow” devem ser reescritas em termos que o estudante já tenha conhecimento, algo como uma

[13] No original: “As the notional machine used in a particular class need only be consistent for the material presented at the time it may be possible to start with a simplified version of a visualization method that gradually adds detail as concepts become more complex in a course or between courses, ensuring that the particular notional machine always meets the requirement of being consistent”.
[14] Disponível em: <https://pythontutor.com/cpp.html> Acesso em: 24 jun, 2025

> noção menos precisa de “espaço disponível”[15] (BOULAY; O’SHEA; MONK, 1981, p. 267, tradução própria).

Mensagem de erro de compilação e execução já são desconstruídas no *backend* para possibilitar a identificação de linha e coluna do local do erro no editor de código no *frontend*. Logo, a identificação das mensagens de erros mais comuns e sua reescrita seriam relativamente simples de implementar no código fonte atual.

Outra melhoria da ferramenta que foi pensada ao longo da produção deste trabalho e reafirmada por sugestões dos estudantes, seria o acréscimo de pequenos textos didáticos em pontos estratégicos como, por exemplo, um texto explicando o que são endereços de memória no primeiro contato do usuário com sua representação.

Por outro lado, textos digitais integrados com visualizações são uma possibilidade em SVPs como o Python Tutor, que podem ser integrados em livros didáticos digitais ou em páginas *web* com apenas uma única linha de JavaScript (GUO, 2013). Isso poderia ser uma funcionalidade a ser explorada no SVP proposto por este trabalho. Sorva, Karavirta e Milma (2013) frisam a importância dessa difusão de ferramentas de visualização:

> Um esforço também poderia ser feito no sentido de produzir materiais didáticos em que interfaces de visualizações de programas se integrem naturalmente com textos e outros conteúdos de tal maneira que usar visualizações se torne uma parte natural do “processo de leitura” do estudante.[16] (SORVA; KARAVIRTA; MILMA, 2013 p. 54, tradução própria).

Por fim, foi notada a ausência de literatura sobre esforços para tornar SVPs acessíveis para pessoas com deficiências. Guo (2013) faz um breve comentário sobre a possibilidade de gerar áudios descrevendo os estados de execução para estudantes com deficiências visuais, mas nada além disso aparece em outros trabalhos encontrados pelo autor.

---

[15] No original: “Error messages are a crucial part of the commentary and they form an important window into the machine. [...] Thus if it is decided that the novice need not be aware of a stack, say, then stack overflow error messages should be reworded in term of some other notion he has been told about, such a some less precise notion of ‘store size’”.

[16] No original: “Work could also be done in order to produce learning materials in which program visualization interfaces naturally with texts and other content so that using visualizations becomes a natural part of the learner’s ‘reading process’”.

No desenvolvimento da ferramenta apresentada por este trabalho, esforços foram feitos nesse sentido ao implementar o *frontend* seguindo as normas do HTML semântico e no uso de atributos WAI-ARIA quando necessário, pensando sempre na experiência por pessoas que navegam por teclado ou com o uso de leitores de tela. Uma importante questão não resolvida seria como representar as setas que conectam uma célula de memória a outra para leitores de tela.

A ferramenta também foi utilizada por uma pessoa cega que conseguiu navegar bem pela página *web*, mas que relatou grande dificuldade no modo de execução: nem a linha destacada no código e nem as mudanças no estado da memória são anunciadas para o leitor de tela.

8 CONCLUSÃO

Diante da evidente dificuldade de estudantes de cursos de computação no Brasil e ao redor do mundo em compreender o conceito de ponteiros e todos seus aspectos profundamente relacionados à memória do computador, faz-se necessário estratégias pedagógicas que consigam suprir essa demanda.

É imprescindível entender que essa dificuldade de compreensão não surge somente da introdução de novos símbolos sintáticos ou novos tipos de variáveis, mas também da incapacidade do estudante em executar, visualizar e compreender o comportamento de máquinas abstratas em seus modelos mentais de maneira consistente e correta.

Este trabalho, então, fundamentado na visão construtivista da teoria de aprendizagem experiencial de Kolb (2015) e no ferramental teórico fornecido pelo conceito de *notional machines* e suas visualizações, oferece como possibilidade de solução um serviço de software interativo para auxiliar o aprendizado de ponteiros em aulas e estudos pessoais.

O sistema de software, munido com uma visualização capaz de representar *notional machines* que são utilizadas direta ou indiretamente por docentes, foi introduzido a turmas de ingressantes no curso de Ciência da Computação na UFRN e obteve, em sua maioria, impressões positivas.

Os usos da ferramenta em sala de aula tanto por estudantes como pelo professor se integraram de maneira satisfatória as aulas e resultaram em uma série de sugestões e críticas à ferramenta que agregaram significativamente para o objetivo exploratório deste trabalho, possibilitando o futuro desenvolvimento de ferramentas e estratégias pedagógicas cada vez melhores e pensadas nas dificuldades dos estudantes.

Os erros experienciados pelos estudantes também indicam um grande potencial para melhoria da ferramenta ao nível de código, a fim de se tornar mais resistente a falhas, escalável e robusto.

8.1 Trabalhos futuros

Estudos quantitativos aplicados em pares de turmas — uma de controle e outra utilizando a ferramenta proposta deste trabalho — seria mais um passo em

determinar, com mais precisão, o impacto do software de visualização de ponteiros no entendimento de estudantes acerca do conceito de ponteiros, gestão de memória e até mesmo conceitos mais gerais como tipos de passagem de parâmetro para subrotinas, escopos de variáveis, funções recursivas, etc.

Já no ponto de vista da qualidade de código da ferramenta proposta, o desenvolvimento extensivo de testes automatizados é recomendado. Testes unitários ajudariam a identificar erros com antecedência nos casos de usos previstos e testes de carga auxiliariam no reconhecimento de problemas de escalabilidade e na sua manutenção.

Portanto, há um grande espaço aberto ao desenvolvimento de pesquisas e aperfeiçoamentos da ferramenta apresentada em diversas perspectivas e escopos diferentes, todos orientados para a busca incessante por formas de complementar ou aprimorar práticas de ensino no campo da computação, visando sempre a formação de estudantes capazes de aplicar os conceitos e as práticas aprendidas independentemente do caminho que escolham seguir em suas vidas.

REFERÊNCIAS

SORVA, Juha; KARAVIRTA, Ville; MALMI, Lauri. A Review of Generic Program Visualization Systems for Introductory Programming Education. **ACM Transactions on Computing Education**, v. 13, n. 4, p.1-64, 2013.

MILNE, Iain; ROWE, Glenn. Difficulties in learning and teaching programming—views of students and tutors. **Education and Information technologies**, v. 7, p. 55-66, 2002.

CACEFFO, Ricardo et al. An Antipattern documentation about misconceptions related to an introductory programming course in C. *In*: **Technical Report 17-15**. Institute of Computing, University of Campinas, 2017. p. 42.

ARIMOTO, Maurício; OLIVEIRA, Weldrey. Dificuldades no Processo de Aprendizagem de Programação de Computadores: um Survey com Estudantes de Cursos da Área de Computação. *In*: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO (WEI), 27., 2019, Belém. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2019.

LAHTINEN, Essi; ALA-MUTKA, Kirsti; JÄRVINEN, Hannu-Matti. A study of the difficulties of novice programmers. **ACM SIGCSE Bull.**, v. 37, n. 3, p. 14-18, 2005.

PEREIRA, Kleber Camara; SOUSA, Reudismam Rolim de. Dificuldades de aprendizagem e fatores que motivam para o estudo para programação: o que mudou desde os estudos de 2018 no curso de TI da UFERSA, Campus Pau dos Ferros? **RECIMA21-Revista Científica Multidisciplinar-ISSN 2675-6218**, v. 5, n. 4, p. e545151-e545151, 2024.

DICKSON, Paul E; BROWN, Neil C. C.; BECKER, Brett A. Engage Against the Machine: Rise of the Notional Machines as Effective Pedagogical Devices. *In*: CONFERENCE ON INNOVATION AND TECHNOLOGY IN COMPUTER SCIENCE EDUCATION, 25., 2020, Trondheim: **Anais** [...]. New York, USA: Association for Computing Machinery, 2020.

FREIRE, Paulo. **Pedagogia do oprimido**. 65 ed. Rio de Janeiro/São Paulo: Paz e Terra, 2018.

KOLB, David A. **Experiential learning**: Experience as the source of learning and development. 2 ed. Estados Unidos da América: Pearson Education, Inc., 2015.

BOULAY, Benedict Du; O'SHEA, Tim.; MONK, John. The black box inside the glass box: presenting computing concepts to novices. **International Journal of Man-Machine Studies**, v.14, n. 3, p. 237–249, 1981.

BEN-ARI, Mordechai. Constructivism in computer science education. **ACM SIGCSE Bull.**, v. 20, n. 1, p. 45-73, 1998.

SORVA, Juha. Notional machines and introductory programming education. **ACM Transactions on Computing Education**, v. 13, n. 2, p. 1-31, 2013.

RAD, Babak Bashari; BAHTTI, Harrison John; AHMADI, Mohammad. An Introduction to Docker and Analysis of its Performance. **IJCSNS International Journal of Computer Science and Network Security**, v. 17, n. 3, p. 228, 2017.

PETAZZONI, Jérôme. Using Docker-in-Docker for your CI or testing environment? Think twice. **jpetazzo**, 2020. Disponível em: https://jpetazzo.github.io/2015/09/03/do-not-use-docker-in-docker-for-ci/. Acesso em: 15 maio 2025.

YAMASHITA, Koichi et al. Classroom Practice for Understanding Pointers Using Learning Support System for Visualizing Memory Image and Target Domain World. **Research and practice in technology enhanced learning**, v. 12, n. 1, p.17-16, 2017.

KUMAR, Amruth N. Data Space Animation for Learning the Semantics of C++ Pointers. **Association for Computing Machinery**, v. 41, n. 1, p. 499–503, 2009.

GUO, Philip J. Ten Million Users and Ten Years Later: Python Tutor's Design Guidelines for Building Scalable and Sustainable Research Software in Academia. **Association for Computing Machinery**, p. 1235–1251, 2021.

GUO, Philip J. Online python tutor: embeddable web-based program visualization for cs education. In: **Proceeding of the 44th ACM technical symposium on Computer science education**. 2013. p. 579-584.

APÊNDICE A - PRIMEIRO FORMULÁRIO DE PONTEIROS

1 - Qual a saída do código a seguir?

```cpp
#include <iostream>
using namespace std;

int main() {
    int x = 5;
    int *y = &x;
    int z = *y;

    cout << *y << ", " << z << endl;

    x = 7;
    cout << *y << ", " << z << endl;

    *y = 2;
    cout << x << ", " << z << endl;

    return 0;
}
```

Resposta esperada (variações de espaços, quebras de linhas e vírgulas foram aceitas como corretas):

5, 5

7, 5

2, 5

2 - Qual a saída do código a seguir?

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 2;
    int b = a * a;
    int *k = &a;
    int c = *k * *k;
    *k = b * *k * c;

    cout << a << ", " << b << ", " << c << endl;
    return 0;
}
```

Resposta esperada (variações de espaços, quebras de linhas e vírgulas foram aceitas como corretas):

32, 4, 4

3 - Qual a saída do código a seguir?

```cpp
#include <iostream>
using namespace std;

int main() {
    int arr[] = {1, 2, 3};
    int *p = arr;

    p++;
    *p *= 2;

    int soma = arr[0] + arr[1] + arr[2];
    cout << soma << endl;

    return 0;
}
```

Resposta esperada:

6

4 - Qual a saída do código a seguir?

```cpp
#include <iostream>
using namespace std;

int main() {
    int arr[] = {1, 2, 3};
    int *p = arr + 2;

    cout << *p * 2 << endl;

    return 0;
}
```

Resposta esperada:

8

APÊNDICE B - SEGUNDO FORMULÁRIO DE PONTEIROS

B.1 Primeira seção do formulário

1 - Escreva um programa que lê dois valores inteiros T e V e cria um array (alocado dinamicamente) de tamanho T e com valores V.

2 - Escreva um programa que lê um valor inteiro T e uma sequência de T valores inteiros. Seu programa deve alocar dinamicamente um array de tamanho T, cujos valores correspondem ao dobro dos valores da sequência lida.

3 - Escreva um programa que lê um valor inteiro N e cria dinamicamente uma matriz identidade de tamanho NxN.

B.2 Segunda seção do formulário

Avaliação da ferramenta no **uso pessoal**

Avaliação da ferramenta "Como ponteiros funcionam" (website).

Responda o seu grau de concordância com as afirmações a seguir, levando em conta somente seu uso **direto** da ferramenta.

1 - "A ferramenta me ajudou a entender o que estava acontecendo no meu programa"

2- "A ferramenta me ajudou a escrever código correto"

3 - "Achei a ferramenta intuitiva e fácil de usar"

4 - "Usaria a ferramenta novamente"

Caso tenha respondido "discordo totalmente" ou "discordo parcialmente" para a afirmação anterior (4), explique o porquê.

Críticas e sugestões

O que você sentiu falta?; Algum empecilho no seu uso?; Pensa em alguma sugestão para melhoria? Responda pensando somente no **uso da ferramenta direto por você**

B.3 - Terceira seção do formulário

Avaliação da ferramenta no **uso pelo professor**

Avaliação da ferramenta "Como ponteiros funcionam" (website).

Responda o seu grau de concordância com as afirmações a seguir, levando em conta somente seu uso em **sala de aula pelo professor, junto aos slides**

1 - "A ferramenta me ajudou a entender o conceito de ponteiros"

2 - "A ferramenta me ajudou a entender passagem de parâmetros em funções e escopo de variáveis"

3 - "A ferramenta me ajudou a entender a diferenças entre *heap* e *stack* (pilha)"

Críticas e sugestões

Achou algo confuso? Algo pode ser melhorado? Responda pensando somente no **uso da ferramenta pelo professor**