UNIVERSIDADE FEDERAL DO PAMPA

Guilherme Souza Santos

SiNo: Uma ferramenta didática de apoio para o ensino de Máquina NORMA

Alegrete
2019

Guilherme Souza Santos

SiNo: Uma ferramenta didática de apoio para o ensino de Máquina NORMA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof. M.ª Letícia Gindri

Coorientador: Prof. Dr. Rafael Santos Coelho

Alegrete
2019

Guilherme Souza Santos

SiNo: Uma ferramenta didática de apoio para o ensino de Máquina NORMA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Trabalho de Conclusão de Curso defendido e aprovado em 28 de novembro de 2019
Banca examinadora:

Letícia Gindri

Prof. M.ª Letícia Gindri
Orientador
UNIPAMPA

Rafael Santos Coelho

Prof. Dr. Rafael Santos Coelho
Coorientador
UFRGS

Alice Fonseca Finger

Prof. Dra Alice Fonseca Finger
UNIPAMPA

Prof. Dr Marcelo Caggiani Luizelli
UNIPAMPA

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

AGRADECIMENTOS

A Deus por ter me dado saúde e força para superar as dificuldades.

A minha família, em particular a minha esposa Daiane e filha Bastet, pelo amor, incentivo e apoio incondicional.

A minha orientadora Professora Letícia Gindri e co-orientador Professor Rafael Santos Coelho, pelo suporte, correções e incentivos.

Aos colegas e amigos que fiz na UNIPAMPA, em particular a Naihara, pelas conversas, discussões e momentos de descontração.

A esta universidade e seus servidores, em particular aos docentes que eu tive a oportunidade de conviver, agradeço pelas críticas, sugestões e discussões durante as aulas pois, contribuíram na minha formação acadêmica.

A todos que direta ou indiretamente fizeram parte da minha formação, em particular, a todo o povo brasileiro que paga os seus impostos, o meu muito obrigado.

RESUMO

O que conhecemos hoje por Teoria da Computação começou a ser formulado a partir do ano de 1900. Neste ano, uma lista de problemas matemáticos foi apresentada por Hilbert e um deles, conhecido como *Entscheidungsproblem*, motivou diversos estudos que contribuíram significativamente para a construção do conceito de computação moderna. Os trabalhos desenvolvidos nos anos seguintes, apresentaram propostas de vários modelos de máquinas de computar, a saber: Máquinas de Turing, Máquinas de Post, Máquinas de Registradores, λ-Cálculo e Autômatos com Pilhas.

Tendo como base o conceito de máquinas de registradores, em particular, a máquina NORMA, este trabalho visa desenvolver um simulador para essa estrutura que possa ser utilizado como ferramenta didática em componentes curriculares de computabilidade ou teoria da computação. Pesquisando em literatura apropriada encontramos alguns simuladores de máquina NORMA dos quais comentamos um pouco na seção 2.4. Entre eles, o que mais se destaca é o simulador desenvolvido pelo Prof. Rodrigo Machado (UFRGS). O diferencial da nossa proposta é justamente tentar facilitar o máximo possível a sintaxe das estruturas a serem programadas no simulador a fim de tornar mais simples a experiência do usuário.

A fim de alcançar esse objetivo, utilizamos procedimentos da Engenharia de Software, começando pela definição do modelo de desenvolvimento de software a ser utilizado, tendo sido selecionado o processo Cascata. Esta escolha justifica-se pois este modelo sugere uma abordagem sequencial e sistemática e, além disso, os requisitos são bem definidos e estáveis. Com isso, iniciamos pelo levantamento de requisitos e em seguida realizamos a fase de planejamento, na qual definimos o cronograma e verificações de status de desenvolvimento. Na etapa de modelagem fizemos a análise e o projeto, seguindo para codificação e por último, foram realizados testes de código e com usuários onde obtivemos alguns feedbacks com relação ao software e suas funcionalidades, concluindo assim o processo. Além disso, foram utilizadas técnicas de qualidades para garantir que o simulador desenvolvido tivesse a qualidade e usabilidade adequadas.

O simulador foi desenvolvido na linguagem Python e também foi desenvolvida uma interface para interação do usuário, contendo as funcionalidades presentes no software. Destacamos que a ferramenta oferece suporte para programar por meio de estruturação iterativa, sendo assim, a maioria das estruturas internas são testes condicionais e laços de repetição, visando facilitar a forma de programar, bem como otimizar o tempo gasto para isso.

Palavras-chave: Simulador. Máquina NORMA. Máquinas Universais.

ABSTRACT

The theory of computation, as we know it today, began to flourish as an independent research field in the early 1900s. One of its main enablers was David Hilbert's famous *Entscheidungsproblem*, a mathematical problem in logic that can be phrased as the following question: is there an effective method that can determine, in a finite amount of steps, the validity of a given formula expressed in the language of first-order arithmetic? In an effort to solve Hilbert's *Entscheidungsproblem*, a group of mathematicians and logicians (most notably Alonzo Church, Alan Turing, Kurt Goedel, Emil Post and Stephen Kleene), throughout the 1920s and 1930s, set out to find a precise, rigorous definition of the intuitive notion of "effective method". As a result of their work, different (but equivalent) formal models of computation were proposed, such as Turing machines, the untyped λ-Calculus and the general recursive functions. Later on, in the 1960s and 1970s, another strand of models of computation was introduced in the literature, namely the unlimited register machines. The NORMA machine (Number-theORetic Register MAchine), a very minimalistic kind of unlimited register machine, was invented by Richard Bird in 1976.
In this work, we present *SiNo*, a user-friendly simulator for the NORMA machine. We hope that this simulator can be used as a didactic tool to help teachers communicate, in a clearer and more straightforward manner, abstract concepts within the scope of the theory of computation. There are some other NORMA machine simulators available on the Internet. However, what distinguises ours from those is that our simulator has a more flexible syntax, which makes the learning experience less tedious.
We have built SiNo using the Python programming language and following the well-known waterfall model of software engineering. During the development of our work, we had the opportunity to employ SiNo in classroom activities and to get positive feedback from students.
Key-words: Simulator, NORMA machine, universal machines

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

CSS Cascading Style Sheets

HTML Hyper Text Markup Language

TCC Trabalho de Conclusão de Curso

LISTA DE SÍMBOLOS

$\cap$	Símbolo matemático para *Interseção*
$\cup$	Símbolo matemático para *União*
$\forall$	Símbolo matemático, leia-se *para todo*
∞	Símbolo matemático para *Infinito*
λ	Letra Grega, *Lambda*
$\mathbb{N}$	Conjunto dos números Naturais
ϕ	Letra Grega, *Phi*

SUMÁRIO

1 INTRODUÇÃO

Um dos marcos iniciais na história e no desenvolvimento da computação foi o *Entscheidungsproblem* (expressão em alemão que significa *problema de decisão*). Essencialmente, o *Entscheidungsproblem*, proposto por David Hilbert e Wilhelm Ackermann em 1928, se reduz à seguinte questão: existe algum procedimento bem definido que nos permite determinar, em uma quantidade finita de passos (ou seja, um *algoritmo*), se uma dada sentença ϕ expressa em aritmética de primeira ordem é válida? O maior desafio apresentado pelo *Entscheidungsproblem* era o fato de que, até o começo da década de 1930, ainda não se conhecia nenhuma formalização matemática para o conceito intuitivo de algoritmo.

Com a chegada dos anos 1930, as primeiras propostas de modelos formais de computação começaram a emergir. Quase que simultaneamente, três formulações distintas ganharam notoriedade, a saber o λ-Cálculo de Alonzo Church (CHURCH, 1936), as máquinas automáticas (hoje chamadas de máquinas de Turing) de Alan Turing e as funções recursivas de Kleene, Gödel e Herbrand (KLEENE, 1938). Ao longo das décadas de 1960 e 1970, destacaram-se também modelos de computação baseados em máquinas de registradores (tais modelos foram inspirados pelas arquiteturas de *hardware* da época), os quais abordaremos em mais detalhe nos parágrafos seguintes.

Grande parte dos conceitos e dos resultados teóricos que conhecemos hoje em computação relativos à sistematização do conhecimento fundacional da área surgiram no desenrolar da primeira metade do século XX. A seguir, descrevemos brevemente alguns dos modelos de computação já mencionados no texto:

- **Máquinas de Turing** (TURING, 1936) - é o modelo teórico mais conhecido e mais comumente utilizado para formalização do conceito de algoritmo. É antes de tudo, um modelo que trata apenas os aspectos lógicos (memória, estados e transições) do funcionamento dos computadores digitais;

- **λ-Cálculo** (CHURCH, 1936) - é um sistema formal cujos termos constituintes podem ser definidos sintaticamente a partir de regras de abstração e aplicação de funções e semanticamente por meio do conceito de reduções. Cabe salientar que o λ-Cálculo influenciou massivamente a criação de linguagens de programação funcionais;

- **Máquinas de registradores** (SHEPHERDSON; STURGIS, 1963) - são modelos baseados na estruturação da memória de acesso. Possuem forte relação com os computadores modernos, sendo esse o objetivo de desenvolvimento dos autores. Dentre os modelos baseados em registradores, destacamos a Máquina NORMA (Number Theoretic Register Machine), proposta por Richard Bird (BIRD, 1967), que é um tipo particular de máquina de registradores e desempenha um papel importante neste trabalho.

Conforme já dito, máquinas de registradores, em particular a máquina NORMA (modelo ao qual daremos mais ênfase) foram elaboradas anos mais tarde em relação ao modelo de Turing. Segundo seus autores, sua criação representa uma tentativa de aproximar a computação moderna, no que se refere à organização da memória e de modelos teóricos.

Informalmente, uma máquina de registradores se baseia nos seguintes conceitos:

- Registradores: unidades discretas de armazenamento capazes de armazenar um número natural (arbitrariamente grande);
- Testes: funções que mapeiam um valor armazenado em um registrador em um valor-lógico (ou seja, verdadeiro ou falso). Usualmente, testamos se o valor corrente de um registrador é igual a 0;
- Operações: *subtração* - subtrair uma unidade do valor de um registrador e *soma* - somar uma unidade ao valor de um registrador;
- Saltos condicionais: podem ou não alterar o fluxo de execução de um programa.

Em linhas gerais, este trabalho tem por objetivo elaborar e propor um meio através do qual, após se familiarizar com os conceitos referentes à máquina NORMA, o aluno possa exercitar e verificar sua compreensão do conteúdo na prática através da codificação de programas em NORMA. Para tanto, vamos desenvolver um simulador para a máquina NORMA.

1.1 Objetivo geral

Desenvolver uma aplicação desktop que seja capaz de simular o comportamento da máquina NORMA - SiNo, utilizando técnicas de desenvolvimento de software abordadas em componentes curriculares do curso de Engenharia de Software.

1.2 Motivação

Este trabalho é motivado pelo interesse em explorar, de forma prática, os conceitos trabalhados em componentes curriculares de Teoria da Computação, mais especificamente, o conteúdo relacionado à máquina NORMA. De modo geral, a abordagem prática nos componentes curriculares supracitados ocorre de forma manual ou por meio de outros simuladores pré-existentes, que não incluem mecanismos compatíveis com todos os cenários que desejamos contemplar. Revisitaremos essa discussão na Seção 3.

Um software capaz de simular o comportamento da máquina NORMA pode proporcionar aos discentes de cursos de computação a aplicação dos conhecimentos adquiridos e a possibilidade de criação de situações-problema com estruturas que manualmente seriam muito trabalhosas ou mesmo impraticáveis de especificar.

Neste sentido, um dos diferenciais deste trabalho é proporcionar um ambiente no qual, estruturas complexas e de difícil execução de forma manual (como a codificação de Gödel - seção 3.4, por exemplo) sejam exploradas e além disso, tornar tais estruturas simples de serem implementadas (no que se refere a sintaxe), o que diminuiria consideravelmente a curva de aprendizado, tornando mais prático a compreensão de tais estrutura com o auxílio do SiNo.

1.3 Organização deste trabalho

Esse trabalho foi organizado da seguinte forma:

- No **Capítulo 2**, introduzimos a formalização dos conceitos necessários para a compreensão da proposta deste trabalho (máquinas, programas e tipos de estruturas de programas), definimos formalmente a máquina NORMA e apresentamos alguns exemplos de programas.
- No **Capítulo 3**, abordamos o simulador de máquina NORMA construído.
- No **Capítulo 4**, descrevemos o plano de teste utilizado para validar o simulador e apresentamos alguns resultados.
- No **Capítulo 5**, expomos e discutimos alguns dos resultados obtidos e comentamos direções de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, definimos dois dos conceitos mais fundamentais deste trabalho: o conceito de máquina e o conceito de programa para uma máquina. Na Seção 2.1, discutimos a noção de programa. Na Seção 2.2, apresentamos os tipos possíveis de estruturações de programas, sendo que um deles será mais enfatizado neste trabalho. Por fim, na Seção 3.3, definimos o que vem a ser uma máquina e definimos a máquina NORMA.

2.1 Programas

Começamos definindo o conceito de programa:

Definição 2.1. (DIVERIO, 2011) [**Programa**] Um programa consiste em um conjunto estruturado de instruções que capacitam um modelo de máquina a aplicar sucessivamente certas operações básicas e testes em uma parte determinada dos dados iniciais fornecidos até que esses dados tenham atingido uma forma desejável.

Ao referenciarmos operações e testes, é praxe usarmos identificadores. Um identificador ou rótulo é uma cadeia finita (uma palavra não vazia) formada por símbolos (normalmente letras e números) oriundos de algum alfabeto previamente escolhido. Por exemplo, em linguagens de programação usuais, o nome de uma variável ou de um comando é um identificador.

2.2 Estruturações de programas

Há várias maneiras possíveis de estruturar instruções de um programa. Neste trabalho, abordamos três estruturas distintas: a estrutura monolítica, a estrutura iterativa e a estrutura recursiva. Na seção seguinte, apresentamos as definições formais dos tipos de estruturas mencionadas, juntamente com exemplos para ilustrar melhor as características de cada uma.

Conforme dito, um programa nada mais é do que a descrição de uma sequência de instruções formatadas de acordo com alguma estruturação pré-fixada. Introduzimos e discutimos nas subseções seguintes as três estruturações consideradas ao longo deste trabalho:

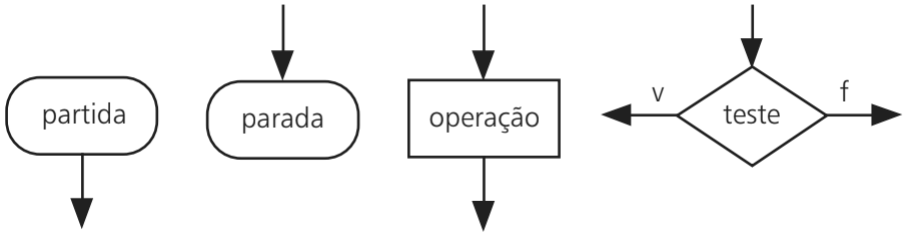
- *Estrutura monolítica*: baseia-se em desvios condicionais e incondicionais; não possui suporte (explícito) à recursão e a comandos de iteração;
- *Estrutura iterativa*: dispõe de mecanismos de repetição (iteração) de trechos do programa; não permite desvios incondicionais;
- *Estrutura recursiva*: possibilita a especificação de sub-rotinas recursivas; não permite desvios incondicionais.

2.2.1 Estrutura monolítica

Programas com estruturação monolítica (ou simplesmente programas monolíticos) podem tipicamente ser representados de duas formas: via *fluxogramas* ou via *instruções rotuladas*.

Uma maneira bastante usual de se especificar um programa monolítico é por meio de fluxogramas. Um fluxograma é um recurso visual que nos permite expor os elementos constituintes básicos do fluxo de execução normal de um programa. Tais elementos constituintes são: exatamente um ponto de *partida* da execução, zero ou mais pontos de *parada* da execução, *operações* e *testes*. Tais elementos constituintes costumam ser identificados de maneira gráfica conforme mostra a Figura 1. A imagem em questão exibe cada elemento do fluxograma e a ação que ele impõe ao fluxo.

Figura 1 – Elementos de um fluxograma para programas monolíticos.



Fonte: Diverio (2011, Teoria da Computação p. 37)

A partir da composição de tais elementos constituintes podemos descrever qualquer programa monolítico indicando, por meio de setas, o fluxo das operações e testes.

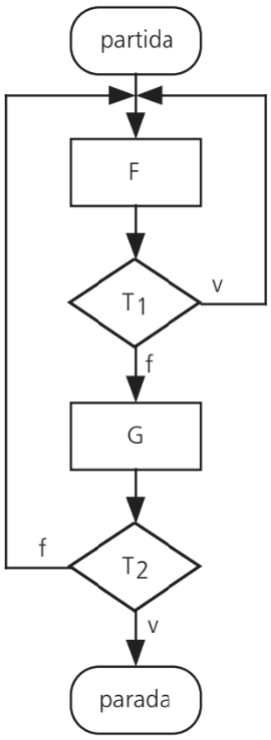
Exemplo 1. Na Figura 5, mostramos um exemplo de um fluxograma. Na imagem em questão, vemos que, após cada teste, existem sempre dois caminhos possíveis. Caso verdadeiro ou falso, o fluxograma executa algum conjunto de ações.

Uma segunda alternativa de representação de programas monolíticos consiste no uso de instruções rotuladas, conforme definimos a seguir:

Definição 2.2. (DIVERIO, 2011) [**Programa monolítico com instruções rotuladas**] Sejam Operações e Testes conjuntos de identificadores de operações e testes, respectivamente. Um programa monolítico P com instruções (provenientes de Operações e Testes) rotuladas é um par ordenado $P = (I, r)$ em que:

- I é um conjunto finito não-vazio de instruções rotuladas e não existem duas instruções diferentes com um mesmo rótulo;
- r é um rótulo inicial que identifica a instrução rotulada inicial em I;

Figura 2 – Programa monolítico representado como um fluxograma.



Fonte: Diverio (2011, Teoria da Computação p. 38)

- um rótulo é dito rótulo final se é referenciado por pelo menos uma instrução de I, mas não rotula qualquer instrução de I;
- cada instrução em I ou é uma operação (desvio incondicional) do tipo

$$R : faça\, O\, vá\_para R'$$

(onde R e R' são rótulos e $O \in$ Operações é um identificador de uma operação) ou é um teste (desvio condicional) do tipo

$$R : faça\, T\, vá\_para\, R'\, senão\, vá\_para\, R''$$

(onde R, R' e R'' são rótulos e $T \in$ Testes é um identificador de teste).

Exemplo 2. Suponha que queiramos representar o programa monolítico descrito no fluxograma da Figura 2 por meio de instruções rotuladas. O programa equivalente pode ser descrito da seguinte maneira:

1. *faça F vá_para* 2
2. *se T*1 *então vá_para* 1 *senão vá_para* 3
3. *faça G vá_para* 4
4. *se T*2 *então vá_para* 0 *senão vá_para* 1

2.2.2 Estruturação iterativa

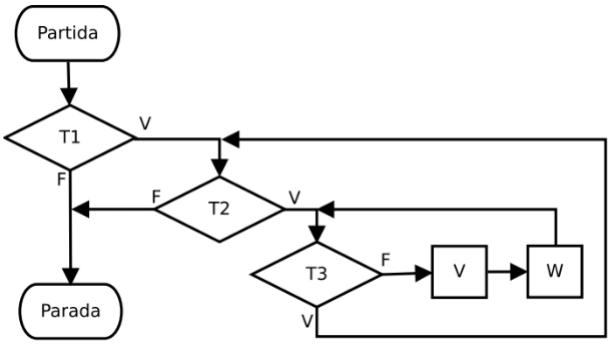
Um programa é dito estruturado de forma iterativa quando suas instruções são formatadas de acordo com os seguintes mecanismos composicionais:

- *composição sequencial* (;): permite a concatenação de dois programas iterativos;
- *composição condicional* (se-então-senão): permite desvios condicionais no fluxo de execução;
- *composição iterativa* (enquanto-faça) e (até-faça): permite repetições controladas de trechos específicos do programa.

Definição 2.3. (DIVERIO, 2011) [**Programa iterativo**] Sejam Operações e Testes conjuntos de identificadores de operações e testes, respectivamente. Seja Iter o conjunto de todos os programas iterativos com instruções pertencentes a Operações e Testes. Então definimos Iter como o conjunto minimal tal que:

- ✓$\in$ Iter (onde ✓denota o "programa iterativo vazio", isto é, o programa iterativo que não faz nada);
- se $O \in$ Operações, então $O \in$ Iter: uma única operação é um programa iterativo;
- se V, $W \in$ Iter, então $(V; W) \in$ Iter: se V e W são programas iterativos, então a concatenação em série de V com W também é um programa iterativo;
- se $V, W \in$ Iter e $T \in$ Testes, então (*se* T *então* V *senão* W) $\in$ Iter
- se $V \in$ Iter e $T \in$ Testes, então (*enquanto* T *faça* V) $\in$ Iter
- se $V \in$ Iter e $T \in$ Testes, então (*até* T *faça* V) $\in$ Iter

Figura 3 – Programa monolítico.



Fonte: Diverio (2011, Teoria da Computação p. 42)

Exemplo 3. O fluxograma exibido na Figura 3 pode ser apresentado usando a estrutura iterativa da seguinte maneira (artificialmente introduzimos uma certa indentação para melhorar a legibilidade do programa):

($se\ T1$

$então$ ($enquanto\ T2$

$faça$ ($até\ T3$

$faça$ (V; W)))

$senão\ V$)

2.2.3 Estruturação recursiva

O terceiro e último tipo de estruturação de programas é a estruturação recursiva.

Definição 2.4. (DIVERIO, 2011) [**Expressão de subrotina**] Sejam Operações, Testes e Procedimentos conjuntos de identificadores de operações, testes e procedimentos respectivamente. Seja Expressões o conjunto de todas as expressões de subrotinas que podem ser definidas a partir de identificadores de instruções e procedimentos pertencentes a Operações, Testes e Procedimentos. Então definimos Subrotinas como o conjunto minimal tal que:

- ✓$\in$ Expressões (onde ✓denota uma "expressão de subrotina vazia", isto é, uma expressão de subrotina que não faz nada);
- se $O \in$ Operações, então $O \in$ Expressões: uma única operação é uma expressão de subrotina;

- se $R \in$ PROCEDIMENTOS, então $R \in$ EXPRESSÕES: um único procedimento é uma expressão de subrotina;
- se V, $W \in$ EXPRESSÕES, então $(V; W) \in$ EXPRESSÕES: se V e W são expressões de subrotina, então a concatenação em série de V com W também é uma expressão de subrotina;
- se $V, W \in$ EXPRESSÕES e $T \in$ TESTES, então $(se\ T\ então\ V\ senão\ W) \in$ EXPRESSÕES

A partir da noção de expressões de subrotina, podemos definir o que vem a ser a estruturação recursiva (um programa recursivo):

Definição 2.5. (DIVERIO, 2011) [**Programa recursivo**] Sejam $R1, R2, \ldots, Rn \in$ PROCEDIMENTOS identificadores de procedimentos. Sejam $E1, E2, \ldots, En \in$ EXPRESSÕES expressões de subrotina. Um programa recursivo P tem a seguinte forma:

P é E_0 onde

$R1\ def\ E1$,

$R2\ def\ E2$, ...

$Rn\ def\ En$

Exemplo 4. O programa apresentado abaixo é um exemplo de um programa recursivo (nele estamos supondo que $R, Z \in$ EXPRESSÕES, $F, G \in$ OPERAÇÕES e $T_1, T_2 \in$ TESTES):

P é $(R; Z)$ onde

$R\ def\ (F;\ (se\ T1\ então\ R\ senão\ (G; S)))$,

$S\ def\ (se\ T2\ então\ \checkmark\ senão\ (F; R))$

$Z\ def\ (se\ T1\ então\ G\ senão\ F)$

2.3 Máquina, função computada e máquina NORMA

Informalmente, entendemos como máquina qualquer dispositivo hipotético dotado de alguma estrutura de memória, equipado com operações (comandos internos que possivelmente alteram o estado da memória) e testes (comandos internos que mapeiam estados da memória em booleanos, ou seja, true ou false) e com interfaces de entrada e saída.

De modo geral, um modelo matemático que se proponha a descrever uma máquina deve ser capaz de suprir os elementos necessários à execução de um programa por tal máquina, tais quais:

- especificação da organização da memória;
- entrada e saída de dados;
- operações e sua semântica;
- testes e sua semântica.

Tendo isso em mente, apresentamos a seguir o conceito de máquina.

Definição 2.6. (DIVERIO, 2011) [**Máquina**] Uma máquina é uma 7-upla

$$M = (\textsc{Mem}, \textsc{Ent}, \textsc{Saída}, \pi_{\textsc{Ent}}, \pi_{\textsc{Saída}}, \Pi_O, \Pi_T),$$

onde

- Mem é o conjunto de valores ou estados de memória;
- Ent é o conjunto de valores de entrada;
- Saída é o conjunto de valores de saída;
- $\pi_{\textsc{Ent}} : \textsc{Ent} \to \textsc{Mem}$ é a função de entrada, isto é, a função que "carrega" um valor de entrada na memória da máquina;
- $\pi_{\textsc{Saída}} : \textsc{Mem} \to \textsc{Saída}$ é a função de saída, isto é, a função que "transfere" um valor de memória para a saída da máquina;
- $\Pi_O : \textsc{Operações} \to (\textsc{Mem} \to \textsc{Mem})$ é a função de interpretação de operação, isto é, trata-se do componente da máquina que especifica precisamente como cada operação suportada age sobre um estado da memória da máquina;
- $\Pi_T : \textsc{Testes} \to (\textsc{Mem} \to \{true, false\})$ é a função de interpretação de teste, isto é, trata-se do componente da máquina que associa a cada teste suportado uma função booleana definida sobre os estados da memória da máquina.

De acordo com Diverio (DIVERIO, 2011), programas e máquinas são tratados como entidades distintas e complementares, sendo ambas necessárias para a definição de computação.

Como já dito, modelos de computação baseados em máquinas de registradores têm por objetivo se aproximar mais com arquiteturas de hardware mais modernas e, por este motivo, máquinas de registradores são fundamentalmente diferentes dos modelos de computação propostas ao longo da década de 1930, como máquinas de Turing. A seguir, formalizamos a definição da máquina NORMA.

Definição 2.7. (DIVERIO, 2011) [**Máquina NORMA**] Seja REG um conjunto contável e infinito de identificadores (tal conjunto pode, informalmente, ser entendido como um conjunto de palavras sobre um alfabeto contendo letras e números). Daqui em diante, interpretamos um elemento R de REG como sendo o "nome" de (ou uma referência a) um registrador da máquina NORMA. A máquina NORMA é definida formalmente como a 7-upla NORMA $= (\text{MEM}^N, \mathbb{N}, \mathbb{N}, \pi^N_{\text{ENT}}, \pi^N_{\text{SAÍDA}}, \Pi^N_O, \Pi^N_T)$ onde:

- O conjunto de valores de memória (os "estados" da memória) de NORMA é o conjunto MEM^N definido como sendo o conjunto de todas as funções $f : \text{REG} \to \mathbb{N}$, isto é, MEM^N é o conjunto de todas as atribuições possíveis de valores em $\mathbb{N}$ aos registradores em REG;
- O conjunto dos valores de entrada de NORMA é o conjunto dos números naturais $\mathbb{N}$;
- O conjunto dos valores de saída de NORMA também é o conjunto dos números naturais $\mathbb{N}$;
- A função de entrada $\pi^N_{\text{ENT}} : \mathbb{N} \to \text{MEM}$ de NORMA é a função que "carrega" um valor de entrada (um número natural dado) em um "lugar específico" da memória de NORMA, a saber o registrador referenciado pelo identificador X. Em outras palavras, π^N_{ENT} inicializa o registrador X com o valor de entrada fornecido e inicializa com 0 todos os demais registradores. Mais formalmente, para todo $n \in \mathbb{N}$, $\pi^N_{\text{ENT}}(n) = h_n$, onde $h_n \in \text{MEM}$, ou seja, $h_n : \text{REG} \to \mathbb{N}$ é a função tal que $h_n(X) = n$ e $h_n(R) = 0$ para todo $R \in \text{REG}$ com $R \neq X$;
- A função de saída $\pi^N_{\text{SAÍDA}} : \text{MEM} \to \mathbb{N}$ de NORMA é a função que, a partir de um "estado" da memória da máquina, "descarrega" para a saída o valor corrente armazenado em um "lugar específico" da memória de NORMA, a saber o registrador referenciado pelo identificador Y. Mais formalmente, dada uma atribuição de valores aos registradores de NORMA, ou seja, dada $f \in \text{MEM}$, $\pi^N_{\text{SAÍDA}}(f) = n$ se, e somente se, $f(Y) = n$;
- A função de interpretação de operações Π^N_O da máquina NORMA concede à máquina NORMA suporte às seguintes operações:
  - para todo $R \in \text{REG}$, Π^N_O interpreta o identificador inc_R como sendo a função que soma exatamente uma unidade ao valor corrente armazenado no registrador referenciado pelo identificador R;
  - para todo $R \in \text{REG}$, Π^N_O interpreta o identificador dec_R como sendo a função que subtrai exatamente uma unidade do valor corrente armazenado no registrador referenciado pelo identificador R (caso o valor corrente armazenado em R seja 0, dec_R não faz nada; nesse caso, dizemos que a operação de subtração "satura" em 0).

- A função de interpretação de testes Π_T^N da máquina NORMA concede à máquina NORMA suporte aos seguintes testes:
  - para todo $R \in \text{REG}$, Π_T^N interpreta o identificador $zero_R$ como sendo a função que testa (retorna true ou false) se o valor corrente armazenado no registrador referenciado pelo identificador R é igual a 0.

Definição 2.8. (DIVERIO, 2011) [**Função computada por um programa**] Seja P um programa para máquina NORMA (em qualquer uma das estruturações vistas anteriormente), a função computada por P, denotada por $\langle P \rangle : \mathbb{N} \to \mathbb{N}$, é a função que mapeia valores de entrada para a máquina NORMA (ou seja, números naturais) em valores de saída da máquina NORMA (ou seja, também números naturais) de acordo com a execução de P em NORMA. Em outras palavras, para todo $n \in \mathbb{N}$, definimos $\langle P \rangle(n)$ da seguinte maneira:

- $\langle P \rangle(n)$ é indefinido caso o programa P entre em *loop* infinito para o valor de entrada n;
- $\langle P \rangle(n)$ é definido como o valor armazenado no registrador Y (o registrador de saída) da máquina NORMA no momento que a execução de P para.

A título de exemplo, a seguir mostramos e comentamos alguns programas, em diferentes estruturações, que podem ser escritos para a máquina NORMA. O leitor irá notar algumas mudanças sintáticas (auto-explicativas) nos identificadores de operações e testes. Decidimos adotar tais mudanças a fim de tornar os programas mais legíveis.

Exemplo 1. (**Programa monolítico**) O seguinte programa monolítico computa a função $n \mapsto n + 1$, isto é, a função que mapeia a cada número natural n seu sucessor $n + 1$. Repare que nas linhas 1–3 temos um laço que transfere, por meio de consecutivos decrementos em X e incrementos em Y, o valor inicialmente contido no registrador X (o registrador de entrada) para o registrador Y (o registrador de saída) e, por fim, faz um último incremento em Y.

1. *se* $X = 0$ *então vá_para* 4 *senão vá_para* 2
2. *faça* $X := X - 1$ *vá_para* 3
3. *faça* $Y := Y + 1$ *vá_para* 1
4. *faça* $Y := Y + 1$ *vá_para* 0

Exemplo 2. (**Programa iterativo**) O seguinte programa iterativo computa a mesma função que o programa monolítico mostrado acima no Exemplo 1.

$$(até\ X = 0$$

$$faça\ (X := X - 1; Y := Y + 1));$$

$$Y := Y + 1$$

Observe que, apesar da simplicidade dos exemplos apresentados, a versão monolítica pode ser mais custosa para se escrever.

2.4 Trabalhos relacionados

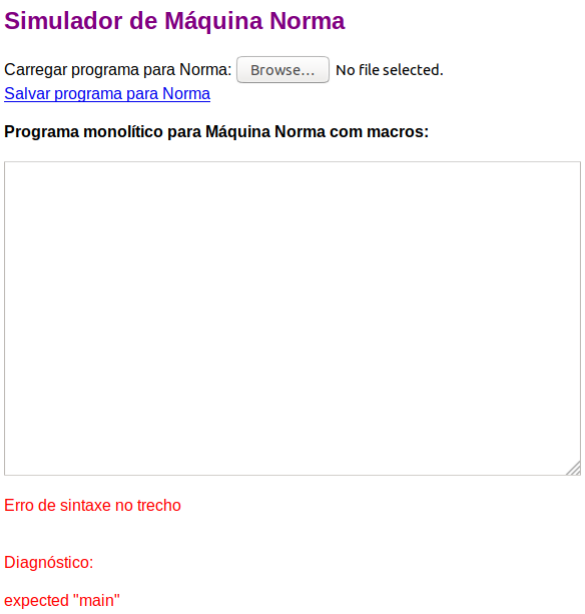
Existem atualmente alguns simuladores de máquina NORMA. Um destes simuladores, cuja interface pode ser vista na Figura 4, foi desenvolvido pelo Prof. Rodrigo Machado (UFRGS) na linguagem funcional ELM[1], tendo suporte para *Web*. O simulador em questão oferece suporte a programação feita de forma monolítica (a estruturação monolítica será abordada na Seção 2.2). Após escrita do código-fonte, o usuário pode compilar o programa e, em seguida, é encaminhado para uma tela onde se encontram várias opções de controle; tal tela pode ser visualizada na Figura 5.

Realizamos alguns testes e foi possível identificar que as seguintes operações estão disponíveis ao usuário:

- Editar programa - ao clicar no botão “editar programa”, o usuário retorna a Figura 5, onde é possível editar o código-fonte;
- Entrada - campo onde se fornece uma entrada para o programa;
- Resetar - ao clicar no botão “resetar”, será limpo o conteúdo dos registradores;
- Passo - ao clicar no botão “passo”, é executada a instrução da linha que o usuário estiver e imediatamente será avançado para a linha seguinte. Tal comportamento pode ser visualmente percebido pelo usuário através de uma linha colorida que salta de uma linha para a outra;
- Rodar - ao clicar no botão “rodar”, o programa é executado;
- Parar - ao clicar no botão “parar”, o usuário para a execução do programa;

[1] ELM é uma linguagem fortemente tipada que compila para JavaScript, Hyper Text Markup Language (HTML) e Cascading Style Sheets (CSS).

Figura 4 – Imagens do simulador de máquina NORMA desenvolvido pelo Prof. Rodrigo Machado da UFRGS



Fonte: Machado (07-06-2019, Simulador de máquina NORMA, página disponível na Web)

- Programa Compilado - embora não seja um comando, fornece uma visão dos passos da execução e, ao rodar o programa, fornece um feedback do procedimento que está sendo executado em cada instante, o que serve como debug ou mesmo para compreensão dos passos executados e sua sequência;
- Registradores - Fornece uma visão dos dados armazenados.

Outros trabalhos foram encontrados através de buscas no GitHub. Para saber se os projetos tinham por objetivo desenvolver simuladores para a máquina NORMA, fizemos a leitura do arquivo *Readme*, local onde usualmente encontram-se comentários a respeito de cada projeto que existe nos repositórios. Não sabemos ao certo a finalidade de cada proposta, assim, presumimos que sejam realizados em componentes curriculares que abordam NORMA como um conteúdo a ser trabalhado. É importante ressaltar que tais projetos não oferecem suporte à programação, são, antes de tudo, softwares que simulam o comportamento de forma mais estática, ou seja, possuem elementos em suas interfaces que ao serem clicados/manipulados, executam o comportamento de adicionar, subtrair, entre outros.

Nesse sentido temos interesse em acompanhar o desenvolvimento dos projetos citados, já que, em Engenharia de Software, usamos o termo *concorrentes* para projetos com o mesmo objetivo. Considerando que nas fases seguintes do nosso trabalho queremos realizar testes com usuários, uma análise das interfaces destes simuladores pode nos

Figura 5 – Imagem do programa compilado no simulador de máquina NORMA desenvolvido pelo Prof. Rodrigo Machado da UFRGS

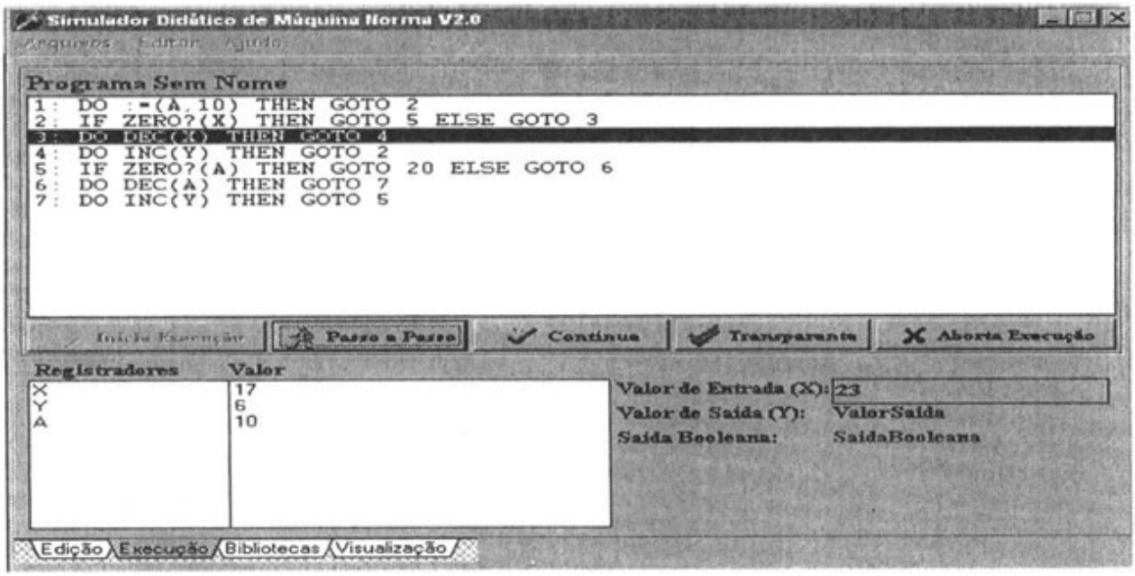
Fonte: Machado (07-06-2019, Simulador de máquina NORMA, página disponível na Web)

ajudar a perceber elementos consistentes para a construção da nossa interface, bem como manter sempre que possível uma padronização e consistência entre os trabalhos aqui citados. Deixamos as referências registradas aqui a título de conhecimento: (VICARI, 2018), (VIANNA, 2018) e (PRADELLA, 2019).

Por último, destacamos que, segundo o autor Diverio (DIVERIO, 2001), existe de um simulador desenvolvido em Delfim, cuja interface, retirada do artigo, pode ser vista na Figura 6. No entanto, não foi possível localizar o simulador nem mesmo indícios de que ainda esteja em uso em algum lugar. Mencionamos tal simulador neste trabalho, pois, ao que tudo indica, foi o primeiro simulador para a NORMA.

Existem outros trabalhos sobre o tema, alguns em páginas Web, sem deixar claro o contexto ou mesmo os objetivos, cujas atualizações esporádicas sugerem que os projetos estejam abandonados. Tais trabalhos não serão citados.

Figura 6 – Imagem do simulador de NORMA em Delfim



Fonte: Diverio (2001, Simulators: Tools for Teaching Theory of Computation p. 489)

3 A FERRAMENTA SINO

Neste capitulo apresentamos a metodologia adotada a fim de desenvolver este trabalho, algumas estruturas internas disponíveis, bem como exemplos de programas em NORMA e como estes mesmo programas podem ser implementados dentro do simulador - SiNo.

3.1 Metodologia

A metodologia adotada foi dividida em quatro etapas:

- Revisão de literatura e estudo dos conceitos básicos;
- Início do processo de desenvolvimento do simulador;
- Escrita do TCC I concomitantemente com uma fase de codificação, validação, correções e testes e
- Escrita do TCC II concomitantemente com realização de mais testes e validação do simulador com cliente.

Na primeira etapa, realizamos uma varredura na literatura disponível sobre o tema com objetivo de compreender os conceitos relacionados à criação do simulador e a estrutura da máquina NORMA. Na seleção das bibliografias consultadas, destacamos o livro (DIVERIO, 2011) bem como alguns materiais de apoio fornecidos pelos professores orientadores (mencionados no decorrer do trabalho). Ainda na primeira etapa, destacamos a exploração do conceito de estruturações de programas (conceito esse que será discutido na Seção 2.2). Observamos que, para o desenvolvimento do simulador em questão, foi indispensável a construção de um Lexer e de um Parser para a sintaxe pretendida dos programas da máquina NORMA. O Lexer (ou analisador léxico) trata da análise de um conjunto de instruções válido e da verificação dos símbolos de uma entrada para checar se fazem parte do alfabeto em questão. O Parser (ou analisador sintático), por sua vez, trata da definição de um conjunto de instruções válidas e da verificação dos símbolos vindos a partir de um Lexer para checar se estão dispostos de tal forma que seja possível executar o comando. Além disso, como conclusão desta etapa, foi realizada uma busca por uma linguagem atual para programar o simulador. Algumas linguagens foram consideradas, a saber Java, Haskell, Python, dentre outras. Em vista dos benefícios e da simplicidade da linguagem, Python foi escolhida em virtude de possuir recursos poderosos em sua biblioteca padrão além de várias bibliotecas auxiliares disponíveis.

Na segunda etapa, ocupamo-nos com o processo de desenvolvimento de software (PRESSMAN, 1992) do simulador com base no modelo em “cascata”, constituído das seguintes fases: i) engenharia de requisitos e projeto com a análise das soluções possíveis e como realizá-la, ii) codificação – boas práticas de programação, documentação do código

e controle de versões presentes no GitHub[1] (SANTOS, 2019) e iii) testes do código-fonte, bem como testes com usuários (planejados e executados no segundo semestre de 2019).

A fase de engenharia de requisitos ocorreu junto ao professor Rafael Coelho, que é coorientador deste trabalho. Os requisitos foram levantados com base na sua experiência ministrando o componente curricular de Teoria da Computação na Universidade Federal do Rio Grande do Sul (UFRGS). Sendo assim, o processo consistiu em questões diretas envolvendo o problema e diálogo direto com cliente. Isso se deu ao fato de que o cliente possuía meios para fornecer diretamente os requisitos, uma vez que o professor tinha experiência e conhecimento das técnicas e meios. Dessa forma, foi necessário apenas documentar e verificar a possibilidade de cada requisito ser implementado no simulador. Não foram definidas restrições de tecnologia ou mesmo de desempenho, apenas uma restrição sobre a plataforma, priorizando o Linux. Foi estabelecido que, se possível, também seria desenvolvida uma versão compatível com Windows. A documentação do projeto, segunda fase do modelo em "cascata", está sendo mantida e atualizada dentro do GitHub/wiki, seguindo boas práticas de engenharia, documentando decisões e mantendo um histórico do processo para a manutenção do projeto. Faz parte do planejamento criar e manter um manual de instalação, descrever e atualizar o histórico de desenvolvimento, correções e melhorias visto que o produto final terá licença livre e, portanto, podem ocorrer colaborações futuras no projeto. Dessa forma, uma documentação atualizada é fundamental.

Na terceira etapa, começamos a fase de codificação do simulador (terceira fase do processo "cascata").

Por fim, na quarta e última etapa prevista foram desenvolvidas as seguintes atividades: i) finalização da codificação, realização de um conjunto de testes finais com código-fonte; ii) validação com cliente e testes com usuários (concluindo as fases do processo "cascata"); iii) definição de quais e como realizar testes com usuários, coleta dos dados gerados a partir dos testes (questionário envolvendo questões de usabilidade); iv) uma nova etapa de correções e de testes (motivadas por sugestões de usuários); v) a escrita do Trabalho de Conclusão de Curso (TCC) II; comparação do simulador desenvolvido neste trabalho com outro simulador conhecido a fim de validar nossa proposta.

3.2 Estruturas Básicas do Simulador

Nesta seção, iremos detalhar e fornecer elementos que ilustram as estruturação do **simulador - SiNo**. Serão apresentadas as estruturas básicas disponíveis no simulador. Destacamos que o intuito do capítulo não é servir de manual de utilização, mas proporcionar uma visão geral do simulador, uma vez que este está disponível no GitHub.

O **SiNo** possui algumas estruturas básicas que permitem, por exemplo, definir registradores, realizar operações básicas e fazer uso de estruturas de controle de forma

[1] GitHub é um sistema de gerenciamento de projeto e versionamento de código utilizado por desenvolvedores.

direta. A seguir são listadas todas as estruturas básicas disponíveis no simulador:

1. Definir registradores, atribuir valores a um registrador;
2. Operações básicas: adição (+), subtração (**-**), multiplicação (*****) e divisão (/);
3. Testes básicos: testar por igualdade (==) e por diferença (**!**=) essas duas possibilidades de testes estamos chamando "R. Básicas". E alguns testes que chamaremos testes avançados: maior (>) e maior ou igual (>=); menor (<) e menor ou igual (<=);
4. Teste condicional sobre os registradores: **if**;
5. Laço de repetição para iterações - já que o simulador implementa uma estrutura iterativa: **while_loop**;
6. Funções ou Macros: são representados por um conjunto de instruções que podem ser repetidas, eliminado assim a necessidade de reescrever várias vezes o mesmo código.
7. Chamada de funções ou calls: é a invocação de uma função previamente definida;
8. Codificação de Gödel (ver Definição 3.1);
9. Arranjos ou vetores;

O leitor deve ter observado que acima foram apresentados mais elementos do que na definição de máquina NORMA (ver definição 2.7). Isto foi feito para atender às necessidades didáticas – de acordo com a engenharia de requisitos, visa atender as necessidades dos clientes (que neste caso são os professores das disciplinas que usarão o simulador). Portanto, a existência de algumas estruturas se deve a questões que estão relacionadas ao processo de ensinar o conteúdo, fazendo com que o conjunto de testes avançados possa ser implementado de forma direta.

Entretanto, um desejo relatado pelos clientes era de que o software tivesse pelo menos dois comportamentos distintos ao executar algum programa. Um deles trata de testes avançados que podem ser habilitados para o uso direto e o outro de testes básicos disponíveis no software. A motivação para essa necessidade de habilitar e desabilitar os testes avançados é mostrar como implementá-los usando operações básicas. No contexto em que os testes avançados estão habilitados, sua implementação seria abstraída pelo conjunto de testes mais diretos e, ainda, aumentaríamos as possibilidades de programação. Ressaltamos ainda que a disponibilidade de tais testes facilitaria a abstração de partes mais importantes em diversos programas, deixando a quantidade de códigos-fonte menor e os testes mais leves tanto para o aluno, quanto para o simulador em si.

Outro ponto a se destacar é a questão relativa às macros (funções). Elas servem para agrupar procedimentos que, em vários programas seriam repetidos, tornando o

programa mais carregado e de difícil compreensão. Desta forma, quando se deseja, por exemplo, somar o conteúdo de dois registradores a um deles, bastaria definir uma macro, e sempre que necessário, fazer a chamada da função (segundo a sintaxe disponível e os dois registradores em questão) não precisando reescrever o código.

3.3 Programas em NORMA

Apresentamos a seguir alguns exemplos de programas para NORMA, ilustrando algumas das estruturas básicas descritas na seção anterior. Os programas a seguir, são exemplos de programas NORMA que podem ser codificados dentro do simulador.

Exemplo 1. Dados os registradores A e B, queremos somar ao registrador A o conteúdo do registrador B. Podemos então escrever um programa NORMA que seja capaz de resolver o problema:

```
1 while B != 0 do
2 |   B = B-1;
3 |   A = A +1
4 end
```

Algoritmo 1: Soma entre A e B, com perda do conteúdo de B

Note que, nesse exemplo, estamos interessados apenas em realizar a soma entre os dois registradores, não queremos ou pretendemos manter o valor que estava em B. Mas, caso fosse necessário, poderíamos então reescrever o programa para que, mesmo após a soma, ainda pudéssemos acessar o conteúdo que estava em B, da seguinte forma:

```
1  REG C = 0;
2  while B != 0 do
3  |   B = B-1;
4  |   A = A +1;
5  |   C = C+1;
6  end
7  while C !=0 do
8  |   B= B+1;
9  |   C = C-1;
10 end
```

Algoritmo 2: Soma entre A e B, preservando o conteúdo de B

Uma possível implementação deste pseudo-código pode ser visto na Figura 7.

Exemplo 2. Considere agora que, dado o registrador A, queremos multiplicar o valor de A pelo número natural 2. Poderíamos escolher outro valor mas, para efeitos de de-

Figura 7 – Implementação do Exemplo 1.

Fonte: O autor

mostração de um pseudo algoritmo, usaremos o valor 2. Teríamos um algoritmo análogo para outros valores.

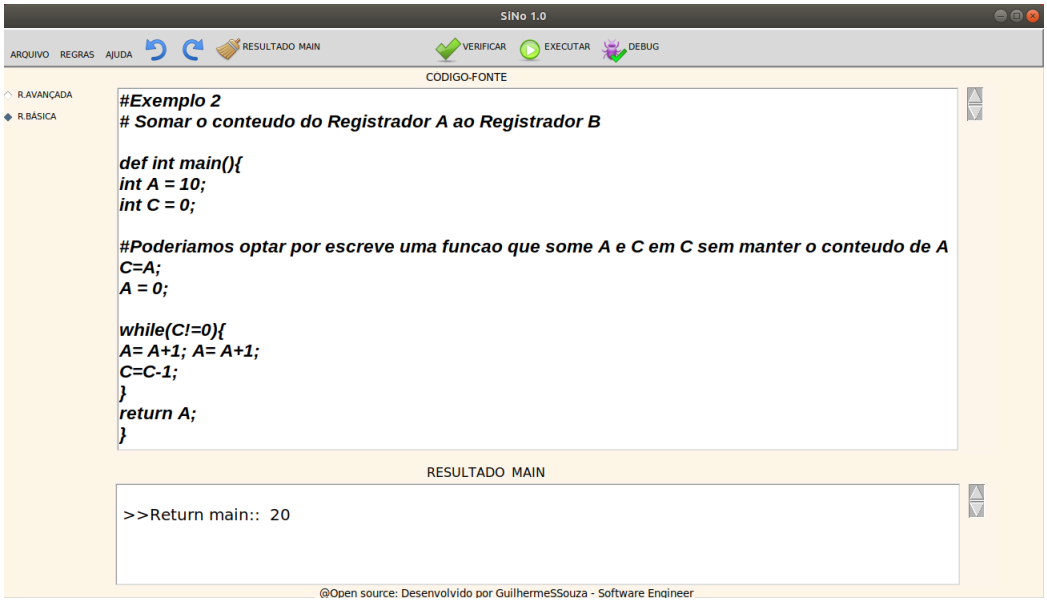
```
1 REG C = 0;
2 C = A;
3 while C != 0 do
4 |   C = C-1;
5 |   A = A +1;
6 |   A = A +1;
7 end
```

Algoritmo 3: Multiplica o valor do conteúdo de A por 2

Note que na segunda linha do Algoritmo 3, especificamente em "$C = A$", estamos supondo que a linha pode ocultar um laço while que atribui o valor que tinha em A a C e zera A – o que seria uma soma com destruição do valor de A. O laço foi abstraído já que o objetivo era multiplicar A por 2 e não necessariamente mostrar a soma entre A e C, como no Exemplo 1. Nossa proposta é que essa abstração seja feita também na prática, ou seja, que o simulador se comporte dessa forma podendo simplesmente usar os comandos "$C = A; A = 0;$" em vez de ter que definir um laço while. Mas, ainda assim, escrever o laço seria uma opção disponível no software.

Dessa forma, o algoritmo abaixo seria executado exatamente como escrito, sem a necessidade de ocultar uma soma de A e C. Uma implementação desse pseudo-código pode ser visto na Figura 8.

Figura 8 – Implementação do Exemplo 2



Fonte: O autor

```
1 REG C = 0;
2 C = A;
3 A = 0;
4 while C != 0 do
5 |   C = C - 1;
6 |   A = A + 1;
7 |   A = A + 1;
8 end
```

Algoritmo 4: Multiplicação do valor de A por 2, usando C

A decisão de implementar o simulador como no exemplo acima teve algumas motivações. Em determinadas situações, a repetição do comportamento "trocar o valor que está em um registrador A para outro registrador B" ou mesmo "ter que zerar o valor que está em um registrador" requer algumas instruções, o que torna a programação lenta e cansativa, sem apresentar ganhos significativos de performance, ou em simplificar o código-fonte.

Por outro lado, ao analisarmos a estrutura utilizada para definir macros (funções), percebemos que, na maioria dos casos, tal estrutura torna mais interessante a programação pois, os comportamentos repetitivos podem ser definidos uma única vez e reutilizados sempre que for pertinente.

Iremos agora mostrar algumas estruturas mais elaboradas que podem ser programadas em NORMA. Dentre essas estruturas, destacamos (a título de exemplo) a codi-

ficação de Gödel e os arranjos. A codificação de Gödel pode ser entendida como uma forma de mapear um par ordenado (x, y) para um número $n \in \mathbb{N}$. Mais precisamente da seguinte forma:

Definição 3.1. [**Codificação de Gödel**] O par ordenando (m, n) é codificado pela função $pair : \mathbb{N}^2 \to \mathbb{N}^+$ definida como:

$$pair(m, n) = 2^m \times 3^n$$

Dado um número natural $n \in \mathbb{N}^+$, podemos decodificar ele em um par ordenado da seguinte forma:

Definição 3.2. [**fst**] A operação de decodificação $fst : \mathbb{N}^+ \to \mathbb{N}$ onde

$$fst(pair(m, n)) = m$$

corresponde à contagem do número de divisões exatas por 2.

Definição 3.3. [**snd**] A operação de decodificação $snd : \mathbb{N}^+ \to \mathbb{N}$ onde

$$snd(pair(m, n)) = n$$

corresponde à contagem do número de divisões exatas por 3.

É importante ressaltar que estamos abstraindo algumas operações como $C/2$ e $C/3$ que na estrutura original da máquina NORMA não existem, sendo necessário que o programador implemente algum conjunto de instruções capaz de realizar a operação. Nossa intenção não é remover essa possibilidade mas, fornecer opções mais simples, tornando mais diretos certos aspectos do programa. Nos exemplos a seguir, apresentamos uma possível implementação para as funções *fst* e *snd*.

Exemplo 3. A decodificação *fst* pode ser representada da seguinte forma:

```
1 REG C = 0;
2 C = B;
3 A = 0;
4 while C divisível por 2 do
5 |   C = C / 2;
6 |   A = A + 1;
7 end
```

Algoritmo 5: Implementando *fst* para o valor em B

Exemplo 4. A decodificação *snd* pode ser representada da seguinte forma:

Para finalizar este capítulo, vamos apresentar a definição de arranjo e uma forma de implementá-lo dentro do simulador. A estrutura em questão é uma lista de dados estruturados, por exemplo $(0, 2, 3, \ldots)$, que representa um arranjo. A fim de trabalhar com tal estrutura temos que definir uma forma de codificar o arranjo e descrever como realizar as operações mais comuns neste tipo de dado.

```
1 REG C = 0;
2 C = B;
3 A = 0;
4 while C divisível por 3 do
5 |   C = C / 3;
6 |   A = A + 1;
7 end
```

Algoritmo 6: Implementando *snd* para o valor em B

Definição 3.4. [**Arranjos**] Sejam $a_1, a_2, \ldots, a_k \in \mathbb{N}$ e $k \in \mathbb{N}^+$. O k-arranjo $(a_1, a_2, \ldots, a_k)$ é codificado como o número natural positivo pela função

$$tuple_k : \mathbb{N}^k \to \mathbb{N}^+$$

definida por $tuple_k(a_1, a_2, ..., a_k) = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_k^{a_k}$, em que $(p_1, p_2, \ldots, p_k)$ são os k-primeiros primos.

Geralmente nesse tipo de estrutura, as operações mais comuns e recorrentes são: obter o valor de uma dada posição i, adicionar e/ou subtrair algum valor k em uma dada posição, codificar um arranjo em um número natural e decodificar um número natural novamente em um arranjo. Para cada uma dessas possibilidades existem alternativas de soluções, uma vez que esta codificação não é a única forma de abordar a estrutura – dado que existem múltiplos registradores – poderíamos também optar por definir que cada posição i do arranjo seja mapeada por um registrador distinto. Se assim for feito, ficará a cargo do programador garantir a consistência da estrutura antes e após as operações básicas.

Para o próximo exemplo, em particular, preferimos usar a definição de arranjos, visto que ela fornece alguns elementos úteis para explorar o entendimento do conceito de soma e subtração existentes sobre os quais se obtém as demais operações.

Exemplo 5. Acessar diretamente o dado em uma posição do arranjo.

```
1 A = 0;
2 C = B;
3 while C divisível por p_k do
4 |   C = C / p_k;
5 |   A = A + 1;
6 end
```

Algoritmo 7: Acesso direto ao valor da posição k do arranjo

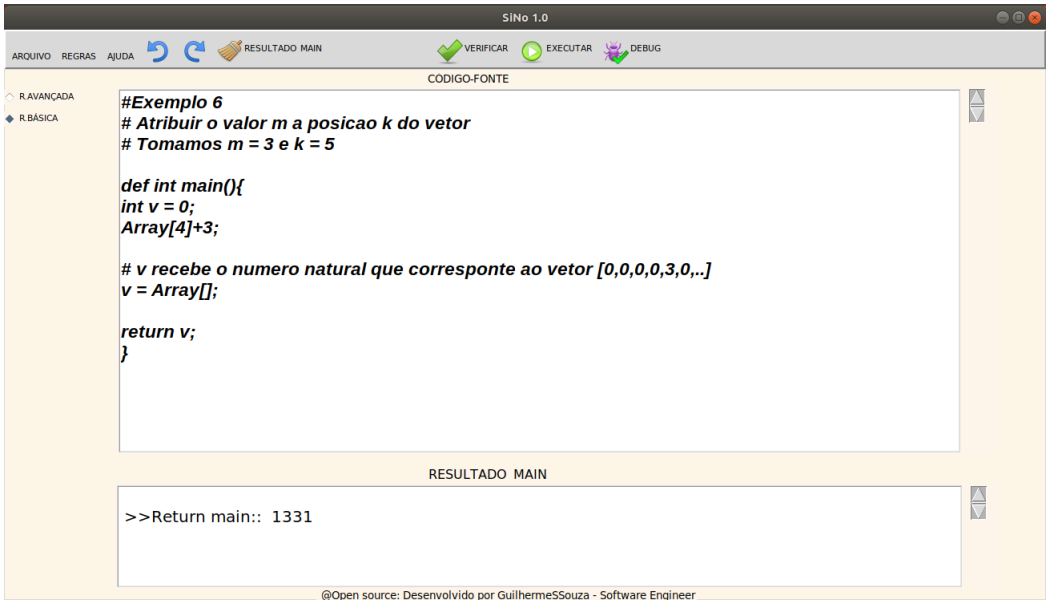
Exemplo 6. Somar um valor m na posição k do arranjo.

Analogamente à soma, podemos definir a subtração em uma posição k de um arranjo. Considerando o exemplo acima, podemos substituir $C = C * p_K$ por $C =$

```
1 A = 0;
2 A = m;
3 C = B;
4 while A != 0 do
5 |   C = C * p_k;
6 |   A = A - 1;
7 end
```

Algoritmo 8: Somar um valor m da posição k do arranjo

Figura 9 – Implementação do Exemplo 6



Fonte: O autor

C/p_k com acréscimo de um teste de divisibilidade antes de executar a "subtração". Uma implementação do Exemplo 6 pode ser visto na Figura 9.

O conjunto de exemplos apresentados aqui ilustram – de maneira breve – a capacidade do simulador que, mesmo construído utilizando operações básicas com números naturais, suporta uma estrutura com complexidade elevada por meio de testes com os valores dos registradores e alguns laços iterativos.

3.4 Estrutura Interna - Desenvolvimento

Nessa seção apresenta-se, ainda que de forma resumida, a estrutura interna da ferramenta, isto é, os seus principais módulos e como eles se relacionam a fim de simular o comportamento esperado. Vale ainda ressaltar que os módulos descritos aqui foram desenvolvidos e testados (teste de código) dentro da fase de " desenvolvimento" do processo cascata, que foi o processo de desenvolvimento de software adotado.

Durante a fase de desenvolvimento do simulador foram desenvolvidos os seguintes

módulos: Módulo Lexer, Módulo Parser e Módulo do IR/Builder que são, em um certo sentido, o coração do simulador produzido. Não discutiremos aqui a motivação que levou a essa escolha frente a outras possibilidades de se implementar o mesmo sistema. Ao invés disso, vamos ilustrar – uma vez feita esta escolha de implementação – qual será a responsabilidade e importância de cada um desses módulos dentro do processo de execução do simulador - SiNo. Dito isso, vamos apresentar uma visão geral do comportamento de cada um dos módulos.

O Módulo Lexer é, informalmente, o responsável por verificar se os caracteres e/ou palavras presentes em um "fluxo" (String de dados) estão escritos corretamente, isto é, se essas " palavras" são aceitas dentro da gramática definida. O Lexer foi codificado em python utilizando a biblioteca PLY (Phyton Lex-YACC). O arquivo que contém as funções necessárias a fim de construir o Lexer possui internamente um conjunto de Tokens válidos (definidos mediante nossa necessidade) e algumas funções, que por meio do uso de Regex, definem um conjunto de nomes ou intervalos de valores aceitos, como por exemplo: números inteiros positivos, nomes para os registradores, entre outros.

A função que inicializa o Lexer recebe um fluxo de dados e – se os dados estão corretamente escritos – os converte em um conjunto de Tokens. Podemos pensar nestes Tokens como um conjunto de objetos em que cada um contém: nome, valor e significado semântico, ordem de escrita segundo o fluxo de string e uma posição dentro do fluxo. Tais informações são utilizadas pelo Módulo Parser (descrito abaixo) para realizar um match com alguma regra definida internamente.

Desenvolvido em python e usando a biblioteca PLY, o Módulo Parser é um mecanismo que possui internamente um conjunto de regras gramaticais pré-definidas com objetivo de interpretar se uma sequência de Tokens corresponde integralmente a alguma regra. O fluxo de Tokens indica ao simulador se ele deve realizar uma soma ou uma subtração e com quais elementos tais operações deve ser realizadas. Além disso, os Tokens também definem em qual momento a operação deve ser executada dentro de um fluxo.

O Módulo Parser recebe um fluxo de Tokens e cria uma árvore de sintaxe abstrata (AST) que pode ser entendida como uma estrutura que armazena as informações sobre o que deve ser executado pelo simulador, por exemplo: se um " nó" deve ser executado primeiro ou após outro, se o "nó " possui algum conjunto de instruções que deve ser executado dentro dele mesmo, dentre outras situações. Além disso, as regras gramaticais presentes no Parser criam, definem e delimitam as estruturas da AST, separando em " galhos" algumas estruturas como: definição e atribuição de valores aos registradores, operações com os registradores, etc.

O último módulo, o IR/Builder, foi desenvolvido em python por meio da biblioteca LLVMlite, e tem por objetivo receber a AST e transformar essa árvore em uma linguagem intermediária de máquina conhecida como IR. A primeira parte do módulo, o construtor da IR, converte a AST em IR. Para isso a biblioteca LLVMlite realiza o processo de

definir e alocar posições de memória, carregar o conteúdo dessas posições, definir e realizar operações sobre essas posições, e ainda, definir e executar os testes e operações descritas no programa. Além disso, a biblioteca utilizada também define o tipo de dado e as operações suportadas por eles, entre outras coisas.

O IR retorna um fluxo de String e, dentro dele, as informações sobre quais posições de memória utilizar, como e quando operar, bem como, em qual posição de memória armazenar esse resultado.

A segunda parte do módulo, o Builder, recebe o fluxo de String que está escrito em linguagem intermediária e transforma esse fluxo em linguagem de máquina que é executado diretamente no computador. Tudo isso é feito por meio da Máquina virtual implementada pela biblioteca LLVMlite, respeitando as características internas dos computadores que executam o simulador, entre outros detalhes de implementação e configurações realizados.

4 TESTES COM USUÁRIOS

Uma das etapas presentes no cronograma do trabalho é verificar e validar o simulador em desenvolvimento. Esta etapa é fundamental no processo de desenvolvimento de software principalmente para medir a aceitação dos usuários e identificar possíveis melhorias e correções. Nesse sentido, elaboramos um teste de aceitação/exploratório a ser aplicado aos usuários finais do sistema. Antes de descrever propriamente o plano de teste e apresentarmos os resultados obtidos, é importante esclarecer que a elaboração das atividades foi norteada por conceitos que são referência na literatura da Engenharia de Software.

Segundo Somerville (SOMMERVILLE, 2011), o teste de aceitação é uma das últimas etapas do processo de validação, geralmente realizado antes da entrega do software ao cliente. Isso se deve ao fato de que, ao realizar este tipo de teste pode-se trazer à tona diversos tipos de erros anteriormente não encontrados, uma vez que é realizado no ambiente final, com dados " reais" que geralmente não foram pensados pelos desenvolvedores/testadores.

Nesse sentido, Presman, divide o teste de aceitação em duas caracterizações:

> **Teste Alfa**: onde o teste de aceitação é executado pelo cliente nas instalações do desenvolvedor. O software é utilizado num ambiente real com a presença do desenvolvedor que registra erros e problemas de uso.
>
> **Teste Beta**: onde o teste de aceitação é realizado pelo usuário final no ambiente de produção, sem a presença ou controle do desenvolvedor. O usuário registra os problemas encontrados e relata-os ao desenvolvedor (PRESSMAN, 1992, p. 167).

Segundo os conceitos apresentados acima, optamos por realizar um teste de aceitação Beta, além de algumas atividades com foco na usabilidade e na exploração do sistema de forma geral.

4.1 Plano de Teste

Nas subseções abaixo apresentaremos os testes com usuários, descrevendo o protocolo utilizado: evidenciando os objetivos, o perfil do usuário para o qual o simulador foi desenvolvido, a lista de atividades e o mecanismo usado para coletar o feedback dos participantes a fim de validar o software.

4.1.1 Propósito do Teste

O propósito deste teste é verificar a performance alcançada pelos participantes e o entendimento das funções presentes no sistema utilizando o protótipo projetado. O objetivo ao aplicar o teste é identificar potenciais alterações/correções necessárias antes da versão final do produto. Para isso, durante a realização das tarefas são identificados erros

e dificuldades envolvendo a utilização do protótipo em atividades rotineiras por meio de um conjunto de atividades previamente definidas. A última etapa do teste é exploratória livre utilizando o simulador a fim de verificar se este se comporta de forma esperada.

4.1.2 Declaração dos Problemas

- Os elementos presentes na interface são intuitivos? Em outras palavras, o simulador é fácil de usar?
- A ajuda presente em "Regras e Help" é eficaz?
- A performance alcançada pelos usuários é a ideal? Em outras palavras, é fácil usar as estruturas para programar?

4.1.3 Perfil do Usuário

Foi definido que seriam necessários pelo menos quatro (4) participantes escolhidos dentre os estudantes do curso de Ciência da Computação da Universidade Federal do Pampa - Campus Alegrete. O número mínimo de quatro participantes é convencionado na literatura por autores como os já mencionados neste capítulo ou mesmo em experimentos que envolvam o tema. Os acadêmicos selecionados devem:

- Possuir familiaridade com a linguagem de programação Python e com o sistema operacional Linux (suficiente para executar arquivos Python pelo terminal Linux).
- Possuir conhecimentos prévios relacionados a Máquina NORMA.
- Conhecer elementos de programação iterativa, comumente utilizadas em linguagens como C, C++, Java, dentre outras.
- O usuário deve possuir conhecimentos básicos relacionados a interação humano-computador a fim de responder o questionário, que pode ser encontrado no Anexo B deste documento.

4.1.4 Metodologia

O teste foi realizado com a finalidade de identificar problemas no simulador. Para isso, foram exploradas as funcionalidades a fim de verificar se ocorriam comportamentos inesperados no software, incluindo questões de usabilidade. A metodologia foi dividida nas seguintes partes:

1. Cada participante sentou-se em um computador com acesso à Web, com o sistema operacional Linux instalado (ou similares), com Python 2.7+ ou superior e acesso ao terminal do sistema.

2. Cada participante recebeu um termo de consentimento onde permitia usar os dados coletados durante o testes, de forma anônima e sem vínculo ao participante. Os dados foram compilados a fim de gerar uma análise sobre a opinião dos usuários. O modelo do Termo de Consentimento e Livre Esclarecimento pode ser encontrado no Anexo A.

3. Cada participante recebeu um script de atividades, que será apresentado na subseção 6.1.5.

4. Cada atividade foi salva em um arquivo e entregue ao avaliador. Este arquivo contém o código executado no simulador, os comentários sobre erros, o feedback e outras sugestões e questões que o participante tenha julgado necessário.

5. Ao fim das atividades, cada participante recebeu um questionário sobre sua experiência com o SiNo 1.0.

4.1.5 Lista de Atividades

Antes de apresentar a lista de atividades, precisamos estabelecer algumas siglas para para compreender cada atividade a ser desenvolvidas pelos usuários durante os testes:

- Restrição da atividade - RT: Significa que a atividade deve seguir alguma especificação, isto é, embora exista várias formas de se resolver a atividade, uma em particular deve ser utilizada.

- Requisitos para executar a tarefa - REQ: O usuário ou sistema deve atender a alguma(s) condição(ões) antes de iniciar a atividade.

- Passos a serem realizados - PR: Conjunto de passos para realizar a atividade.

A seguir são listadas as atividades realizadas durante o teste de usabilidade/exploratório:

1. Atividade: Carregar o arquivo “Exemplo1.txt” para a interface do sistema. Escrever um comentário no arquivo e salvar o arquivo na pasta com código-fonte do sistema.

   a) REQ: Simulador em execução e caixa de código-fonte sem qualquer código escrito.

   b) PR: Usar o carácter # antes da linha de comentário.

2. Atividade: Somar o conteúdo dos registradores a = 9 e b = 7, no registrador a.

   a) REQ: Simulador em execução e caixa de código-fonte sem qualquer código escrito.

   b) PR: Criar uma função main ( ).

c) PR: Criar os registradores, a, b e a instrução a = a + b.

d) PR: Retornar o valor do registrador a.

3. Atividade: Realizar o produto entre os valores dos registradores a = 5 e b = 3 e retornar o resultado da multiplicação. *RT: Realizar a multiplicação sem usar a operação " * ", ou seja a operação deve ser realizada por meio de somas.*

a) REQ: Simulador em execução e caixa de código-fonte sem qualquer código escrito.

b) Criar a função soma; Mesma função da atividade anterior agora com parâmetros (a, b). Retorna a soma de dois registradores;

c) PR: Criar uma função main ( ).

d) PR: Criar uma laço que realize "n" vezes a soma.

e) PR: Retorna o produto.

4. Carregar o arquivo "Erro1.txt" para provocar erros no simulador.

a) REQ: Simulador em execução e a caixa de código-fonte sem qualquer código escrito

b) PR: Trocar o nome da função main ( );

c) PR: Executar o simulador;

d) PR: Corrigir o nome para main ( ) -> Executar simulador

e) PR: Remover o '=' da atribuição do a;

f) PR: Executar o simulador;

g) PR: Corrigir o '=' -> executar o simulador;

h) PR: Comentar o "return c;"

i) PR: Executar o simulador;

j) PR: Remover o comentário de "return c;" - >executar o simulador;

k) PR: Descomentar o if (>=) e o código entre ;

l) PR: Executar o simulador;

m) PR: Trocar o conjunto de regras;

n) PR: Executar o simulador;

5. Atividade Exploratória.

a) REQ: Simulador em execução e a caixa de código-fonte sem qualquer código escrito

b) PR: Exploração livre dos elementos da interface e funcionalidades do sistema;

c) PR: O participante tem liberdade para programar e testar o simulador (Codificar, forçar erros, verificar variações na sintaxe e explorar por comportamentos inesperados);

d) PR: Comentar e salvar o código-fonte que porventura venha a apresentar o comportamento inesperado, além de comentar qual foi o comportamento e como reproduzi-lo e entregar o código-fonte ao avaliador;

4.1.6 Questionário

O questionário foi montado com base na escala Likert, um tipo de escala de resposta psicométrica usada habitualmente em questionários. Esta escala é a mais usada em pesquisas de opinião. Ao responder um questionário baseado nesta escala, os participantes especificam seu nível de concordância com uma afirmação.

Nosso questionário teve como objetivo identificar a opinião dos usuário sobre o quão fácil havia sido usar o simulador para resolver atividades relacionadas à máquina NORMA.

O questionário pode ser encontrado no apêndice B.

Deixamos claro que adotamos a escala Likert – e não outros mecanismos que poderiam gerar dados e, posteriormente, estatísticas – devido ao cenário. Como o componente curricular do qual usaríamos a aula para aplicar o questionário possui oferta anual e turmas com uma quantidade não muito grande de alunos, tínhamos uma baixa expectativa com relação ao número de voluntários que se disponibilizariam a realizar o teste. Se adotássemos um outro método e houvesse uma participação ínfima dos alunos, a fim de tentar validar o simulador, seria necessário refazer os testes buscando mais voluntários, o que demandaria mais tempo. Nesse sentido, prevendo todas as possibilidades, optamos pela escala Likert por representar a opinião dos usuários e ser comumente utilizada nos trabalhos relacionados a Engenharia de Software, onde estamos interessados na satisfação do usuário com o produto final.

4.2 Aplicando o Teste

O experimento contou inicialmente com 17 participantes volutuários porém, antes de iniciar as atividades em questão, 4 dos 17 participantes optaram por não seguir o experimento.

No início do experimento foram apresentados de forma resumida os objetivos do teste, como seriam realizadas as atividades, a necessidade do termo de consentimento e além disso, foi garantido aos participantes que a qualquer momento eles poderiam desistir da atividade sem obrigação de finalizá-la. Ao final do experimento foi solicitado aos participantes que preenchessem o questionário (o preenchimento era voluntário). Foi

informado que os dados seriam usados para mensurar a aceitação do sistema proposto como uma solução didática de apoio ao ensino da cadeira de Teoria da Computação.

Após essa breve explicação sobre a realização do experimento, com cada participante já em sua máquina e executando o sistema, foi apresentada a descrição das atividades e estabelecido um tempo médio de dez minutos para realização de cada uma delas. Ao término das atividades, foram recolhidos os arquivos dos participantes que quiseram realizar o feedback por escrito, bem como o questionário dos que tiveram interesse em respondê-lo. O experimento contou com 13 participantes e todos optaram por responder também ao questionário.

Vale salientar que houve bastante interação entre os participantes e também com o simulador. Aparentemente todos os acadêmicos pareciam bastante interessados em desenvolver as atividades da melhor forma possível e conversavam entre si em diversos momentos a respeito das atividades e do simulador. A experiência foi gratificante já que a proposta visa contribuir com atividades em alguns componentes curriculares e a ferramenta foi desenvolvida para facilitar o aprendizado dos alunos. Essa interação e os feedbacks recebidos demonstraram uma aceitação inicial do simulador até pelo próprio interesse dos alunos em testar e encontrar erros.

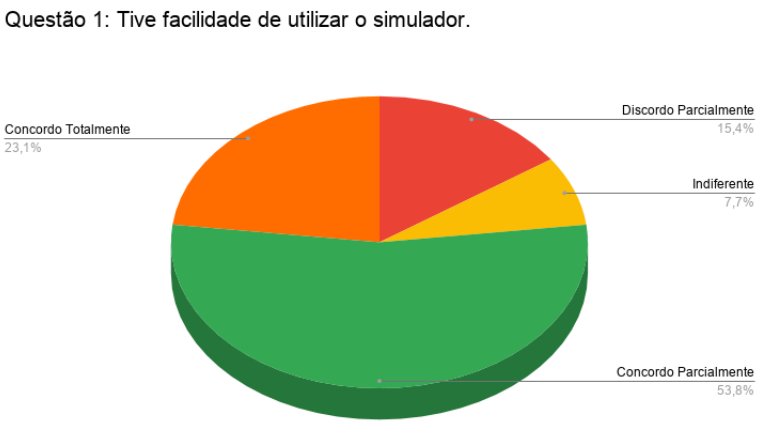
4.2.1 Resultados do Questionário

Selecionamos para apresentar nesta seção algumas questões dentre todas que foram aplicadas no questionário (Anexo B), a fim de ilustrar – de forma geral – nossos objetivos ao realizar o experimento. As respostas dos participantes nos dão uma visão geral da experiência vivenciada. Os gráficos referentes às demais questões poderão ser vistos no anexo A. Embora não sejam diretamente discutidas as contribuições das demais perguntas, é possível encontrar padrões nas respostas, complementando assim o entendimento a respeito da opinião do usuário.

Como parte do objetivo central do experimento, a compilação dos resultados abaixo apresenta um panorama geral sobre a interação dos usuários com o sistema e a forma como eles perceberam o experimento. Desta forma, o tema central das questões (apontado pelos gráficos abaixo) poderia ser sintetizado nos seguintes pontos: A facilidade de usar o simulador – elementos, interface, botões, campos de texto, feedbacks apresentados aos erros de sintaxe, nas atividades guiadas e exploratórias – e o quanto esses elementos ajudam nessa interação.

A primeira questão, "**Tive Facilidade de utilizar o simulador**", obteve 13 respostas, conforme apontado pela figura 10. A opção *Concordo Totalmente* obteve 21,1% do total, que corresponde a 3 respostas, a opção *Concordo Parcialmente* obteve 53,8% do total, que corresponde a 7 respostas, a opção *Indiferente* obteve 7,7% do total, que corresponde a 1 resposta e *Discordo parcialmente* obteve 15,4% do total, que corresponde a 2 respostas.

Figura 10 – Abordando a facilidade em usar o simulador



Fonte: O autor

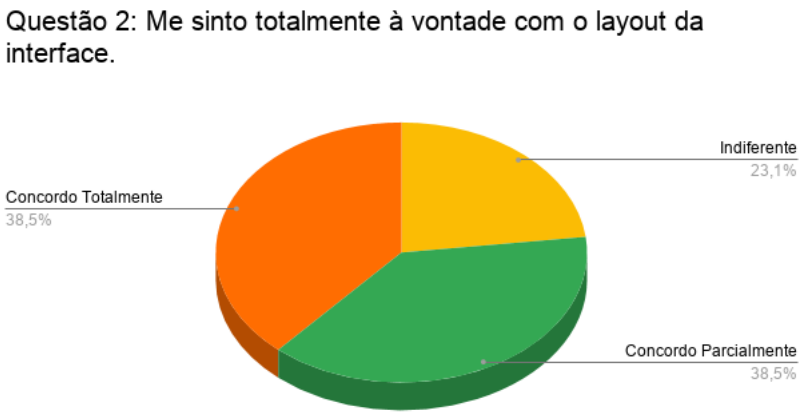
Tendo como base as duas primeiras opções mais votadas *Concordo Totalmente* e *Concordo Parcialmente* (que juntas somam 74,9% do total de respostas dos participantes) e levando em conta o fato de que apenas 2 dentre os 13 tiveram alguma dificuldade mais contundente ao usar o simulador, podemos afirmar que o uso do simulador é fácil e intuitivo. É importante observar que não aconteceu nenhum tipo de treinamento antes da execução do experimento e que dessa forma, aqueles indivíduos que atendem ao perfil escolhido – que também é o perfil padrão dos usuários finais do simulador – tiveram, em sua maioria, uma experiência satisfatória.

A segunda questão, "**Me sinto totalmente à vontade com o Layout da interface**", obteve 13 respostas, conforme apontado pela figura 11. As opções *Concordo Totalmente* obteve 38,5% do total, que corresponde a 5 respostas, *Concordo Parcialmente* obteve 38,5% do total, que corresponde a 5 respostas e *Indiferente* obteve 23,1% do total, que corresponde a 3 respostas.

Quanto a essa questão, se considerarmos o gráfico, a maioria das respostas apontam que o layout e os elementos da interface (apesar de poucos) são de fácil compreensão e comunicam o que querem dizer. Analisando agora os 23,1% que foram indiferentes, podemos dizer que consideraram que mesmo que não fossem significativos os elementos da interface, ao menos eles não comunicaram algo diferente do esperado. O que nos leva a concluir que foi unânime a opinião que os elementos estão claros e objetivos no que se propõem a comunicar.

A terceira questão, "**Os feedbacks apresentados durante minha utilização foram claros e objetivos**", obteve 13 respostas, conforme apontado pela figura 12. As opções *Concordo Totalmente* obteve 38,5% do total, que corresponde a 5 respostas,

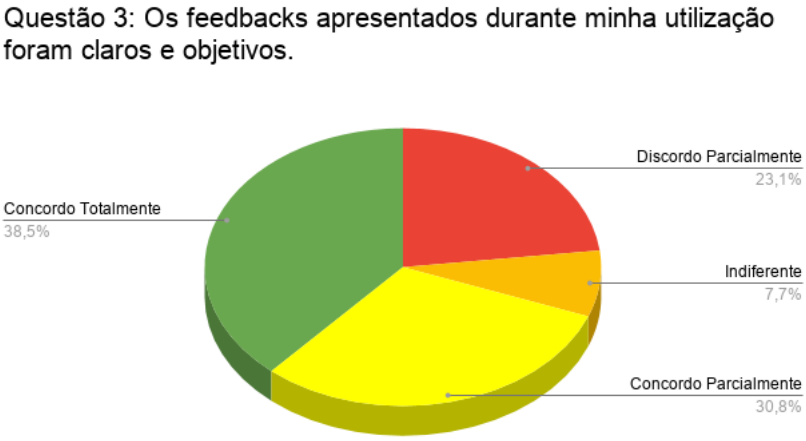
Figura 11 – Abordando o layout da interface



Fonte: O autor

Concordo Parcialmente obteve 30,8% do total, que corresponde a 4 respostas, *Indiferente* obteve 7,7% do total, que corresponde a 1 resposta e *Discordo parcialmente* obteve 15,4% do total, que corresponde a 2 respostas.

Figura 12 – Abordando os feedback apresentados, de forma geral, durante o uso do simulador



Fonte: O autor

Essa questão mostra um ponto importante, ela nos indica que, assim como era esperado, os feedbacks presentes na aplicação são fundamentais para seu uso. Desta forma podemos notar que em um contexto geral, a maioria dos usuários, julgou que os

feedbacks são consistentes.

A oitava questão, "**As mensagens de erro apresentadas me ajudaram a entender a causa do problema**", obteve 13 respostas, conforme apontado pela figura 13. As opções *Concordo Totalmente* obteve 15,4% do total, que corresponde a 2 respostas, *Concordo Parcialmente* obteve 7,7% do total, que corresponde a 1 resposta, *Indiferente* obteve 15,4% do total, que corresponde a 2 respostas, *Discordo parcialmente* obteve 38,5% do total, que corresponde a 5 respostas e *Discordo Totalmente* obteve 23,1% do total, que corresponde a 3 respostas.

Figura 13 – Feedback e Mensagens de erro durante todo o uso do sistema

Fonte: O autor

Como indicam as respostas à questão, os feedbacks dos erros ainda se encontravam em fase de desenvolvimento. Por esse motivo, alguns feedbacks não estavam condizentes com que era esperado, isto é, eram mensagens genéricas e que, em alguns casos, não diziam claramente o conteúdo do erro justamente por serem report de erros da biblioteca utilizada. Após esse etapa, alguns feedbacks de erros foram trabalhados e outros finalizados a fim de atender a expectativa dos usuários quanto a essa questão.

4.2.2 Feedbacks e Report Fornecidos

Como parte do experimento, todos os alunos foram orientados a fornecer feedbacks e/ou reports das atividades realizadas de forma voluntária e livre, sem que os feedbacks tivessem qualquer identificação dos participantes. Desta forma, alguns ofereceram reports com sugestões e/ou informando bugs ou ainda qualquer resultado inesperado, segundo o julgamento do próprio usuário. Salientamos que não necessariamente tais reports seriam

um bug/erro/problema no sistema mas, algo que o usuário não identificou como um comportamento normal.

Como parte do propósito geral do experimento, a lista das principais sugestões e erros passou a incorporar o conjunto de melhorias do sistema em uma escala própria de prioridades. Alguns dos reports eram potenciais bugs, mas não trataremos detalhadamente neste texto sobre isto devido a sua origem técnica e a possível discussão da tecnologia por trás do sistema. Dito isso, segue abaixo uma compilação das principais sugestões e erros observados durante o experimento.

Número	Sugestão
3	Adicionar Atalhos: Ex. Ctrl + Z, Crtl + S, Page Up, Page Down
2	A mensagem de erro poderia conter também a linha onde se dá o erro.
2	Existir um manual do sistema, descrevendo as funções existentes
1	O Scroll do Campo Resultado: Scroll automático
1	Botão/ícone do Menu: Limpar todos os campos de texto
1	Trocar Clear do Menu: Usar New File (Consistência com outros softwares)
1	Uma versão por linha de comando

Dentre essas sugestões, algumas foram implementadas por julgarmos que elas contribuiriam para uma melhoria significativa na experiência com o simulador. Alguns atalhos foram implementados, como Ctrl + z e Ctrl + y, que dado a tecnologia é Shift + Ctrl + z. Os feedbacks foram melhorados de forma geral, com a descrição clara dos motivos que geravam os erros. E, além disso, houve uma melhora no texto presente no GitHub – manual – com uma descrição mais precisa das funcionalidades e sintaxe presentes no simulador.

5 CONSIDERAÇÕES FINAIS

Ao final do processo de desenvolvimento, algumas questões ainda necessitam ser respondidas, tais como:

1. O simulador suporta números naturais até a ordem de 2^{63}, isso pode afetar a experiência do usuário? Embora a grandeza desse número pareça razoável, sabemos que para uma máquina este é um número pequeno. Desta forma, essa condição poderia limitar a experiência dos usuários com o simulador, visto que alguma funcionalidade poderia ter um crescimento numérico relativamente rápido ultrapassando o limite do simulador.

2. As estruturas presentes no simulador são suficientes para suprir as necessidades de ensino já discutidas neste trabalho? Em outras palavras, tais estruturas favorecem o ensino e aprendizagem ou ainda seriam necessárias alterações a fim de tornar o simulador uma ferramenta de ensino que contribua significativamente para o aprendizado?

3. E, por último, podemos nos perguntar quais outras estruturas poderiam ser implementadas no simulador atual para torná-lo mais completo a fim de cumprir com o objetivo de ser uma ferramenta de apoio didático para o ensino dos conceitos relacionados a máquinas NORMA?

Qualquer que seja a resposta para as perguntas acima, neste momento, ela seria incompleta, visto que todas essas questões só poderiam ser respondidas após algum período de utilização do simulador seja com testes para usuários direcionados a responder cada uma destas questões ou com o uso da ferramenta didática em sala de aula. Mesmo que sejam questionamentos válidos, a solução para cada uma demandaria muito tempo e esforço para ser implementada e, ainda assim, não seria possível garantir a sua real contribuição. Não descartamos a possibilidade de que após um período maior de testes com o simulador seja necessário aperfeiçoá-lo.

Outro ponto importante de nota é o fato de que o código sendo aberto, qualquer usuário poderá vir a contribuir para tornar a experiência melhor fazendo alguma sugestão, testando questões relevantes ao simulador ou mesmo realizando implementação de novas funcionalidades ou melhorias nas funcionalidades já existentes.

Ao concluirmos a escrita deste trabalho de conclusão de curso, já tendo realizado o estudo dos conceitos sobre NORMA, a pesquisa em literatura pertinente sobre o tema, a busca por simuladores existentes, a etapa do processo de desenvolvimento do software e os testes com usuários, estamos seguros em afirmar que o simulador é uma ferramenta bastante interessante de apoio didático. Mais especificamente, o simulador poderá contribuir significativamente para o ensino de máquinas NORMA nos componentes curriculares relacionados à Teoria da Computação, visto que, em diversos aspectos, o software simplifica o trabalho e melhora a experiência do usuário, otimizando o tempo gasto com

programação e oferecendo diferentes recursos em relação aos outros modelos citados neste trabalho, ainda que possuam estruturas em comum.

Além disso, determinadas estruturas são inviáveis de serem trabalhadas sem o apoio de uma ferramenta didática. Como exemplo podemos citar as codificações de Gödel, arranjos, entre outros. Seria necessário realizar um grande número de passos manualmente para testar tais estruturas. E ainda, mesmo com o suporte de alguns dos simuladores mencionados, seria necessária uma quantidade elevada de instruções para se obter o resultado desejado. Nesse sentido, um dos grandes diferenciais do simulador aqui apresentado é permitir ao usuário implementar de forma mais direta tais estruturas no simulador.

O desenvolvimento do software foi realizado de forma organizada e gerenciada, utilizando técnicas de qualidade com objetivo de aprimorar a usabilidade e experiência. A ferramenta desenvolvida ao final do processo foi validada por meio de testes com o usuário, onde obtivemos alguns resultados sobre a aceitação e alguns feedbacks com relação ao uso do simulador. A aceitação (inicial) dos usuários, bem como a experiência obtida por meio da utilização foi muito boa, visto que o simulador foi considerado pela maioria dos usuários, simples e fácil de ser usado. Dito isso, reforçamos a ideia que o software é uma potencial ferramenta para contribuir substancialmente para o ensino de máquina NORMA. Portanto, podemos afirmar que os nossos objetivos foram alcançados no decorrer deste trabalho.

5.1 Trabalhos Futuros

Levando em consideração tudo o que já mencionamos neste capítulo, podemos destacar alguns potenciais trabalhos futuros. Como por exemplo:

- Comparar o simulador SiNo com o simulador desenvolvido pelo Prof. Rodrigo (UFRGS) a fim de verificar/mensurar questões envolvendo usabilidade, facilidade em utilizar o software e eficiência como ferramenta de apoio didático dentro dos componentes curriculares já citados neste trabalho.

  Isso poderia ser feito por exemplo realizando um treinamento com relação ao uso de cada um dos simuladores e o desenvolvimento de atividades pré-programadas e também de atividades livres. Poderia ser aplicado um questionário utilizando até mesmo alguma outra escala (dependendo da quantidade de participantes interessados) a fim de coletar dados e posteriormente, a partir deles, mensurar as questões que listamos. Também seria interessante a aplicação de questionários direcionados aos professores com intuito de, na opinião deles, obter um comparativo com relação ao apoio didático fornecido por cada uma das ferramentas e a facilidade em utilizar o software, bem como, verificar o interesse e a possibilidade do SiNo vir a ser adotado em alguma próxima vez que o professor ministrasse a componente.

- Verificar alternativas/estratégias para superar a limitação apontada neste trabalho (lembre que o simulador, neste momento, suporta números naturais até a ordem de 2^{63}). Inicialmente podemos verificar o quanto isto afeta de fato a experiência do usuário, o que pode ser feito, por exemplo, a partir de outros testes envolvendo a ferramenta e/ou trocar a abordagem para com os usuários. Algumas estratégias já começaram a ser investigadas no sentido de superar a limitação, porém nada ainda foi implementado. Uma alternativa seria tentar adaptar algumas estratégias implementadas em algumas linguagens de programação, o que poderia gerar codificações em baixo nível. Outra possível solução seria não utilizar a biblioteca LLVMlite, o que poderia fornecer alternativas para resolver o problema porém, essa decisão é difícil de ser tomada dada a quantidade de retrabalho necessário.

REFERÊNCIAS

BIRD, R. **An Introduction to the Theory of Computation**. [S.l.]: Jhon Wiley, 1967. Citado na página 19.

CHURCH, A. An unsolved problem of elementary number theory. **American Journal of Mathematics**, v. 53, n. 2, p. 345–363, 1936. Citado na página 19.

DIVERIO, T. A. Simulators: Tools for teaching theory of computation. **World Conference on Computers in Education (2001 : Copenhagen)**, p. 483–493, 2001. Citado 2 vezes nas páginas 34 e 35.

DIVERIO, T. A. **Teoria da Computação**. [S.l.]: Bookman, 2011. Citado 10 vezes nas páginas 23, 24, 25, 26, 27, 28, 29, 30, 31 e 37.

KLEENE, S. C. On notation for ordinal numbers. **Journal Symbolic Logic**, n. 3, p. 150–155, 1938. Disponível em: <http://www.thatmarcusfamily.org/philosophy/Course_Websites/Readings/Kleene%20-%20Ordinals.pdf>. Citado na página 19.

MACHADO, R. **Simulador de Máquina Norma**. [S.l.]: Inf.ufrgs, 07–06–2019. <http://www.inf.ufrgs.br/~rma/simuladores/norma.html>. Citado 2 vezes nas páginas 33 e 34.

PRADELLA, P. **TC-MaquinaNorma**. [S.l.]: GitHub, 2019. <https://github.com/evernife/TC-MaquinaNorma>. Citado na página 34.

PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach**. Third. New York: McGraw-Hill, 1992. Citado 2 vezes nas páginas 37 e 49.

SANTOS, G. S. **Simulador Máquina NORMA**. [S.l.]: GitHub, 2019. <https://github.com/GuilhermeSSouza/SimuladorNorma>. Citado na página 38.

SHEPHERDSON, J. C.; STURGIS, H. E. Simulators: Tools for teaching theory of computation. **Journal of the ACM (JACM)**, v. 5, n. 27, p. 1–37, 1963. Citado na página 19.

SOMMERVILLE, I. **Engenharia de Software**. [S.l.]: Presman, 2011. Citado na página 49.

TURING, A. M. On computable numbers, with an application to the entscheidungs-problem. **London Mathematical Society**, v. 10, n. 12, p. 230–245, 1936. Citado na página 19.

VIANNA, P. **Exemplos de Programas em Máquina Norma**. [S.l.]: GitHub, 2018. <https://github.com/patrickvianna/maquina-norma>. Citado na página 34.

VICARI, P. **Norma Machine**. [S.l.]: GitHub, 2018. <https://github.com/pvicari/norma-machine>. Citado na página 34.

Apêndices

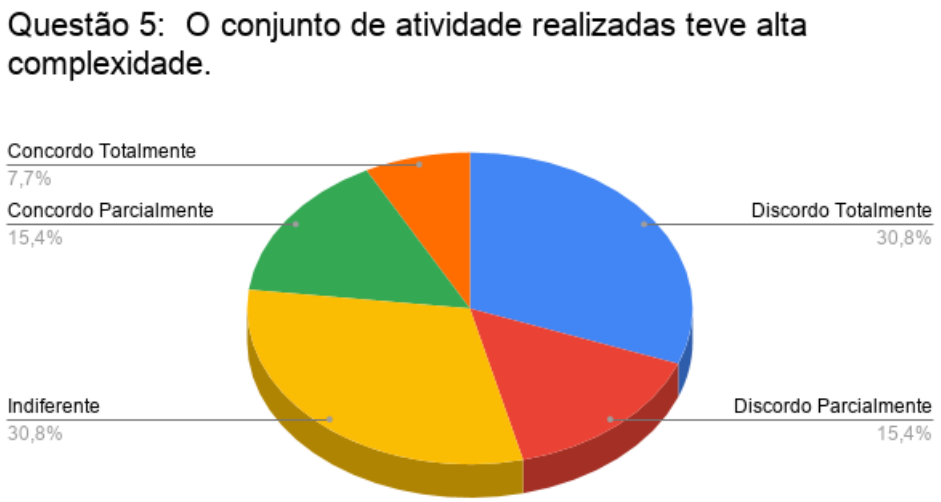
APÊNDICE A – RESULTADO COMPLETO DO QUESTIONÁRIO DE USABILIDADE

Apresentamos agora uma compilação dos dados referentes as questões presentes no questionário do Anexo B que não estão presentes na seção 4.2.1. Não discutiremos as implicações destas questões mas, com algum cuidado, pode-se verificar que esses dados de alguma forma comprovam (ou mesmo validam) os demais apresentados na sessão sobre resultados deste trabalho.

Figura 14 – Os elementos, textos e campos, são claros e corresponderam ao esperado.

Fonte: O autor

Figura 15 – O conjunto de atividade realizada teve alta complexidade

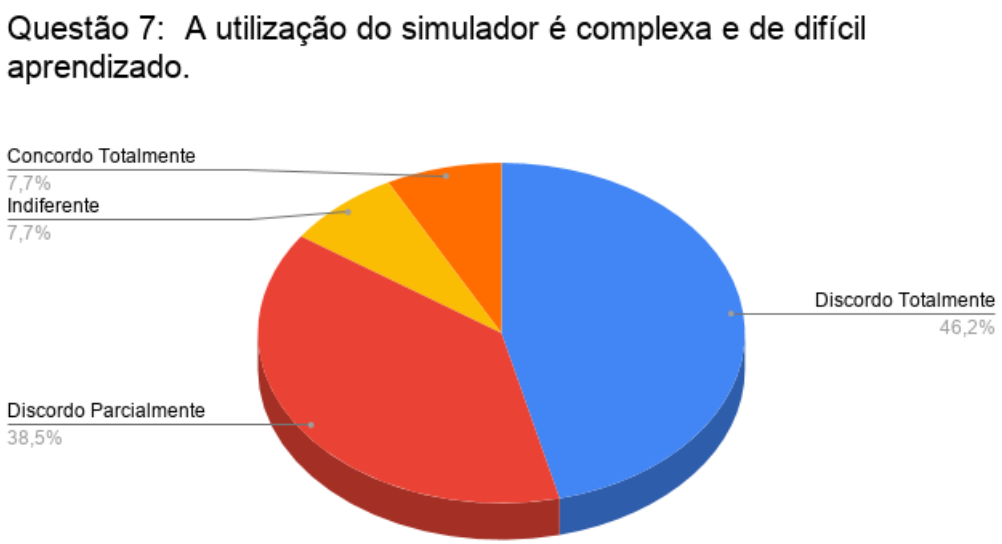


Fonte: O autor

Figura 16 – Me sinto mais confortável em codificar com programação iterativa

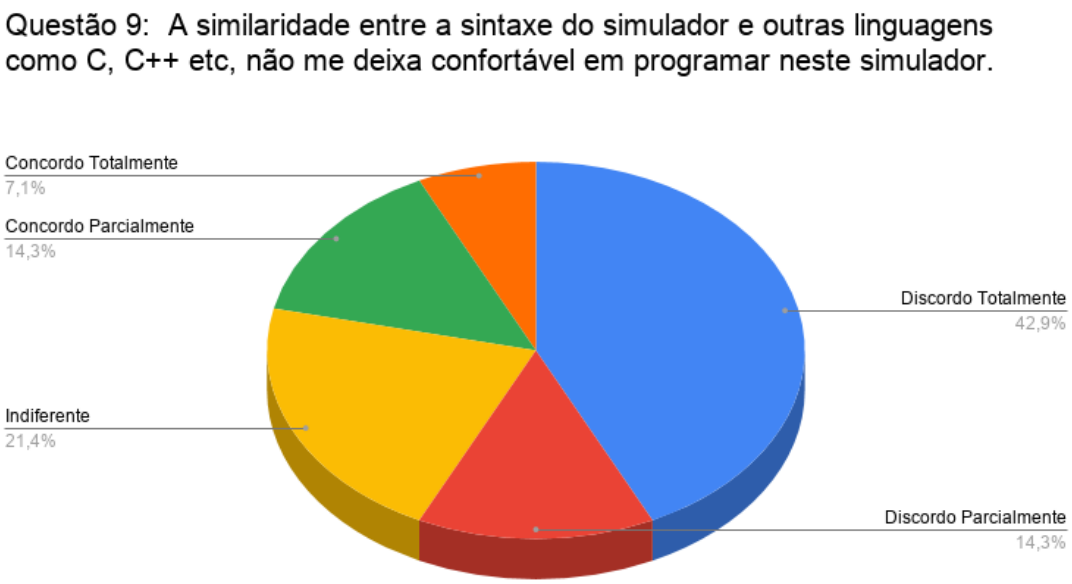
Fonte: O autor

Figura 17 – A utilização do simulador é complexa e de difícil aprendizado



Fonte: O autor

Figura 18 – A similaridade entre a sintaxe do simulador e outras linguagens como C, C++ etc, não me deixa confortável em programar neste simulador.



Fonte: O autor

Anexos

ANEXO A – TERMO DE LIVRE CONSENTIMENTO

TÍTULO DO TESTE DE USABILIDADE: Teste de Usabilidade/Exploratório do Simulador de Máquina Norma - SiNo 1.0 RESPONSÁVEL: Universidade Federal do Pampa - Unipampa – Bacharelado em Engenharia de Software. O Sr.(a) está sendo convidado(a) a participar do teste de usabilidade/exploratório que tem como objetivo avaliar a facilidade de uso, ou seja, a usabilidade do sistema Simulador do modelo de Máquina NORMA - SiNo 1.0. Além do uso exploratório, que tem por objetivo, identificar potenciais problemas de usabilidade e identificar comportamentos inesperados no sistema, salientamos que o estudo tem foco no simulador e seu comportamento, não na capacidade do participante em resolver as atividades propostas.

O teste de usabilidade consiste em atividades em que o participante deverá executar no simulador, seguindo um protocolo de passos pré-estabelecidos e ao final, responder a um questionário fechado sobre os pontos explorados, seguindo a escala de Likert.

A identidade do participante será preservada. Os resultados do estudo serão divulgados em trabalho de conclusão de curso, congressos, publicações científicas ou outras formas de publicações.

A participação é voluntária e pode deixar de ocorrer a qualquer momento, sem que isto acarrete qualquer prejuízo ao participante. Qualquer esclarecimento necessário poderá ser realizado através do contato com Guilherme Souza Santos, autor da pesquisa, tel: (55) 99713-5020, email: guilthys@gmail.com ou com o Profª. Mª. Leticia Gindri, orientadora da pesquisa, pelo endereço Av. Tiaraju, 810 - Ibirapuitã, Alegrete - RS, 97546-550, email: leticiagindri@unipampa.edu.br.

Minha participação está formalizada através da assinatura deste termo.

Participante: _____

_____ , _____ de_____ de 2019.

ANEXO B – QUESTIONÁRIO DE USABILIDADE

O questionário abaixo é baseado na escala de Likert, abordando o tema, facilidade de uso do sistema e, como tal, foi usado a seguinte escala de concordância:

- Escala de concordância:

  1. Discordo totalmente
  2. Discordo parcialmente
  3. Indiferente
  4. Concordo parcialmente
  5. Concordo totalmente

- Questionário utilizado para avaliar a opinião dos usuários durante o experimento:

  1. Tive Facilidade de utilizar o simulador.
  2. Me sinto totalmente à vontade com o Layout da interface.
  3. Os feedbacks apresentados durante minha utilização foram claros e objetivos.
  4. Os elementos, textos e campos, são claros e corresponderam ao esperado.
  5. O conjunto de atividade realizadas teve alta complexidade.
  6. Me sinto mais confortável a codificar com programação iterativa.
  7. A utilização do simulador é complexa e de difícil aprendizado.
  8. As mensagens de erro apresentadas me ajudaram a entender a causa do problema.
  9. A similaridade entre a sintaxe do simulador e outras linguagens como C, C++ etc, não me deixa confortável em programar neste simulador.