Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma avaliação comparativa das linguagens Rust e C para o ensino de programação

Danilo Y. Fukuda

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof. Dr. Edison Ishikawa

Brasília
2021

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma avaliação comparativa das linguagens Rust e C para o ensino de programação

Danilo Y. Fukuda

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Prof. Dr. Edison Ishikawa (Orientador)
CIC/UnB

Prof. Dr. Maria de Fátima Ramos Brandão
CIC/UnB

Dr. Maristela Terto de Holanda
Membro CIC/UnB

Prof. Dr. Wilson Henrique Veneziano
Coordenador do Curso de Computação — Licenciatura

Brasília, 28 de Maio de 2021

Dedicatória

Dedico este trabalho primeiramente a Deus, que sempre me protegeu no trajeto de Formosa à Brasília e que me deu forças para não desistir. Dedico também às pessoas que me influenciaram e me apoiaram todo esse tempo, minha mãe, que sempre esteve comigo e é um exemplo para mim, principalmente nos piores momentos, minha tia Margarida, que sempre investiu em mim, meu vô Morioshi Fukuda, que é um exemplo de homem e me apoiou financeiramente em tudo que pedi e por final, ao professor Ishikawa, que apesar de muitos erros e deslizes meus, sempre me guiou e me apoiou na execução deste trabalho.

Agradecimentos

Eu agradeço primeiramente a Deus, pois vejo que Ele me protege todas as vezes que faço o trajeto de Formosa a Brasília, a todas as pessoas envolvidas diretamente e indiretamente na execução deste trabalho, professores que me influenciaram, principalmente o Prof. Edison Ishikawa, que me orientou muito bem neste trabalho, pessoas da minha família que me apoiam, meus amigos, cada pessoa que diz palavras de apoio para mim, cada uma delas tem um lugar especial no meu coração.

E finalmente dedico a todos que acham que eu estou demorando muito para formar, pessoas que perguntam quando que vou acabar a faculdade, só para olhar pra mim esperando ver algum tipo de sofrimento, pois foi com essas pessoas que aprendi a não ouvir comentários pessimistas e realmente escolher quem anda do meu lado.

Resumo

O principal problema identificado nas disciplinas introdutórias de programação está relacionada a dificuldade de abstração dos problemas e na lógica. Sendo assim a tarefa de encontrar uma linguagem de programação de primeiro contato que facilite o aprendizado dos alunos se torna essencial para o ensino da programação. Rust é uma linguagem de programação que está em ascensão, por ter vários pontos positivos a seu favor na sua utilização, como por exemplo, a segurança de memória é garantida. Neste trabalho será exposto quais artigos que defendem o uso do Rust em turmas introdutórias de programação com base em uma revisão sistemática da literatura e uma comparação da linguagem Rust com a linguagem C, concluindo que Rust é uma alternativa viável para o ensino de programação no lugar de C.

Palavras-chave: Rust, Ensino de Programação, Revisão Sistemática da Literatura

Abstract

The main problem identified in the introductory programming disciplines is related to the difficulty of abstraction of problems and logic. Thus, the task of finding a first programming language that facilitates students learning becomes essential for teaching programming. Rust is a programming language that is on the rise, as it has several positive points in its use, such as memory security is guaranteed. And in this work it will be exposed which articles that defend the use of Rust in introductory programming classes based on a systematic review of the literature and a comparison of the Rust language with the C language. The conclusion is that Rust is a viable alternative to C in a first programming course.

Keywords: Rust, Teaching Programming, Systematic Literature Review

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A disciplina de Algoritmos e Programação de Computadores(APC) da Universidade de Brasília (UnB) é uma disciplina aplicada para cursos que são da área de Computação, que tem como objetivo introduzir os alunos da disciplina à programação, como por exemplo, Ciência da Computação, Computação(licenciatura), Engenharia Mecatrônica, Engenharia de Computação, Engenharia Mecatrônica e Engenharia Elétrica.

A ementa é divida por princípios fundamentais de construção de programas, construção de algoritmos e sua representação em pseudocódigo e linguagens de alto nível, noções de abstração, especificação de variáveis e funções, testes e depuração, padrões de soluções em programação, noções de programação estruturada, identificadores e tipos, operadores e expressões, estruturas de controle: condicional e repetição, entrada e saída de dados, estruturas de dados estáticas: agregados homogêneos e heterogêneos, iteração e recursão, noções de análise de custo e complexidade, desenvolvimento sistemático e implementação de programas, estruturação, depuração, testes e documentação de programas, resolução de problemas, aplicações em casos reais e questões ambientais.

A seleção de uma linguagem de programação para um curso introdutório à programação de computadores sempre foi fundamental, bem como contencioso, tal linguagem é geralmente referido como Primeira Linguagem de Programação (First Programming Language - FPL). O objetivo do primeiro curso de programação de computadores é fornecer conceitos e conhecimentos para os alunos compreenderem o fundamental dos constructos da programação de tal forma que devem ser capazes de programar um determinado problema [1].

Por exemplo, em 1994 Pascal foi usado por 40% nas turmas de programação introdutória, já em 2006 por 0%, e que em 2011 foi observado que Java e C ++ foram os mais usados(79% juntos), Em Portugal, 2016-2017, a sequência de linguagens de programação mais comum nos cursos introdutórios de programação em 46 cursos analisado foi apenas C (48%), seguido por apenas java (22%), C + Haskell (9%), C + java (4%), Scheme +

java (4%). Havia também sequências residuais de Excel + C, apenas Python, Python + HTML + java, Python + java, Scheme + C ++ e XML + java [2]. O que se observa é que com o decorrer do tempo novas linguagens de programação são criadas e passam a ser utilizadas em CS1.

Os professores do Departamento de Ciência da Computação da Universidade de Brasília (CIC) optaram em utilizar a linguagem C na disciplina de APC, pois ela é uma linguagem de programação com alto desempenho e também permite trabalhar com detalhes de mais baixo nível da arquitetura do computador, ambos aspectos necessários às outras disciplinas do curso. Muitas disciplinas do CIC usam a linguagem de programação C como Sistemas Operacionais, Compiladores, Organização e Arquitetura de Computadores, entre outras. Além disso, existe uma percepção de que se consegue ser mais proficiente na primeira linguagem de programação que se aprende.

Pyhton também é uma escolha plausível para ser a primeira linguagem de programação a ser ensinada aos alunos, cuja aprendizagem é mais fácil, porque tem ferramentas poderosas que refletem a maneira como as pessoas pensam e implementam o código [3].

Porém, alguns professores alegam que a linguagem de programação Python tem baixo desempenho e não seria adequados para disciplinas que necessitam de uma linguagem de programação de nível médio, como a linguagem C. No entanto, durantes este período de aulas remotas devido à pandemia da Covid-19, optou-se por usar a linguagem Python no lugar da Linguagem C para facilitar o aprendizado do pensamento computacional, uma vez que houve uma compreensão de que aprender a programar computadores com uma linguagem de sintaxe mais simples eliminaria algumas dificuldades no aprendizado feito de forma remota e frequentemente assíncrona. Provavelmente, após o retorno ao ensino presencial, C seja a linguagem a ser usada.

Por isso o escopo deste trabalho se restringe entre Rust e C, uma vez que Rust é um potencial candidato a substituir a linguagem C quando do retorno às aulas presenciais.

1.1 Identificação do problema

É altamente reconhecido que as dificuldades envolvidas em ensino de programação em um curso introdutório, surgem da complexidade do processo cognitivo que é necessário para desenvolver este habilidade [4]. Com isso os alunos tem que superar suas dificuldades na disciplina de APC num espaço de tempo reduzido, somente um semestre. No entanto, nem todos podem dedicar maior quantidade de tempo extra classe semanal para aprender a programar, o que leva a uma alta taxa de insucesso na disciplina.

1.2 Pergunta de pesquisa

Quais os pontos positivos da utilização da linguagem de programação Rust em um curso introdutório de programação?

1.3 Objetivos

Para responder à pergunta de pesquisa temos como objetivo geral verificar se existem vantagens em se usar a linguagem de programação Rust no lugar de C no ensino de uma primeira linguagem de programação. Para atingir este objetivo tem-se os seguintes objetivos intermediários:

1.3.1 Objetivo específicos

- Fazer revisão sistemática da literatura sobre o uso de Rust no ensino de programação;
- Comparar a linguagem de programação Rust com C na implementação dos conceitos fundamentais de programação utilizados em um primeiro curso de programação de computadores;

1.4 Justificativa

Os resultados obtidos neste trabalho podem vir a auxiliar a disciplina APC, de forma geral, e servir como base para futuras alterações na disciplina, com benefícios para o CIC e para todas as turmas de APC.

Tendo em vista o desenvolvimento da proposta apresentada, organizou-se este trabalho dividindo-o em 5 capítulos, sendo o primeiro capítulo a introdução, que descreve a identificação do problema, a pergunta da pesquisa, objetivos principais e específicos, justificativa e resumo do trabalho.

O capítulo 2 apresenta a fundamentação teórica, que apresenta os principais artigos que ilustram com maior profundidade a identificação do problema, a linguagem C e a Linguagem Rust, e por fim a metodologia usada para este trabalho.

O capítulo 3 apresenta a revisão sistemática da literatura, que conta com 8 etapas, segundo a metodologia de Chitu Okoli para a execução de uma revisão sistemática, porém neste trabalho foi dividido em 4 etapas, fazendo a junção de algumas etapas, que são: A primeira etapa: O objetivo; a segunda etapa: O protocolo, pesquisa e tela prática; a terceira etapa: A avaliação da qualidade e síntese de dados; e a quarta etapa: O processo. Seguindo as 4 etapas foram selecionados 5 artigos que são: "A cursory overview

of the Rust programming language", "A Functional Paradigm using the C Language for Teaching Programming for Engineers", "A Physical Platform For Teaching Distributed Systems In A Simplified Setting", "From Theory to Systems: A Grounded Approachto Programming Language Education"e "Monografia na matéria MAC 5742:Introdução à Computação Paralela e Distribuída Professor Alfredo Goldman Vel Lejbman IME USP".

O capítulo 4 apresenta a comparação entre a linguagem C e Rust, comparando as IDEs utilizadas para criar os códigos e as diferenças nas semânticas de cada linguagem e se tais diferenças seriam favoráveis ao uso da linguagem Rust em matérias introdutórias de programação.

O capitulo 5 apresenta a conclusão do trabalho, que apresenta os pontos fortes da linguagem Rust para sua utilização nos cursos introdutórios de programação, e o que se espera do futuro da linguagem Rust no ensino de programação introdutória e até na sua utilização para programas e sistemas mais seguros.

Capítulo 2

Fundamentação Teórica

2.1 O pensamento computacional

O pensamento computacional é uma forma para resolver problemas, projetando sistemas e compreensão do comportamento humano que se baseia em conceitos fundamentais para computação segundo Jeannette Wing [5] no seu livro "Computational thinking and thinking about computing", o pensamento computacional é um tipo de pensamento analítico que associa-se com pensamento matemático nas formas gerais em que podemos abordar a solução de um problema. Ele se associa também com o pensamento de engenharia nas formas gerais em que nós podemos abordar a concepção e avaliação de um sistema grande e complexo que opera dentro das restrições do mundo real. Se associa também com o pensamento científico nas maneiras gerais pelas quais podemos abordar a compreensão da computabilidade, inteligência, a mente e o comportamento humano. A essência do pensamento computacional é abstração. Na computação, abstraímos noções além das dimensões físicas do tempo e do espaço. Nossas abstrações são extremamente gerais porque são simbólicas, onde as abstrações numéricas são apenas um caso especial.

2.2 As dificuldades no ensino da programação

Um programador experiente necessita de muitas habilidades e muita experiência. Algumas das habilidades necessárias tem uma relevância essencial para o processo de produção código do programa, segundo Jenkins [6], algumas dessas competências necessárias para o processo de produção de código de programa são; problema de capacidade de resolução, alguma ideia da matemática subjacente ao processo são essenciais, um programador deve usar o computador de forma eficaz, deve ser capaz de criar o programa em um arquivo, compilá-lo e encontrar a saída. O programa produzido deve ser testado, e erros encontrados e corrigido. Estas são habilidades fáceis de identificar e consequentemente eles são

abordados na maioria cursos de programação. Porém existem habilidades menos óbvias. Estes podem ser classificadas como "habilidades para a vida". A programação é normalmente ensinada como um assunto fundamental no início de um curso de graduação. Este é um momento difícil para muitos alunos - um momento de transição, como eles se adaptam à vida e estudar na universidade. Eles podem estar vivendo longe de casa pela primeira vez, eles podem ter dificuldades em fazer novos amigos e encontrar os pés em um novo ambiente, e eles podem lutar para chegar a termos com a gestão de suas próprias finanças e sua próprio tempo privado e de estudo. Em meio a isso eles vão estar encontrando alguns dos materiais mais básicos na sua programação. Esse é um material potencialmente desafiador que vai formar a base do resto da sua aprendizagem. Eles estarão perdidos se não entenderem isso. Este pode ser um conteúdo bastante difícil de dominar mesmo quando um aluno está bem resolvido. No entanto, o curto espaço de tempo, apenas um período letivo) para aprender uma nova habilidade só aumenta a dificuldade para estes alunos.

2.3 A linguagem de programação C

Para explicar o porque do uso da linguagem C em cursos de programação introdutória, seria importante mostrar um breve resumo sobre quais linguagens de programação foram usadas em cada determinada década.

O final da década de 1960 e o início da década de 1970 viram uma revolução na "programação estruturada", na qual o controle de fluxo baseado em GoTo de linguagens como FORTRAN, COBOL e Basic deu lugar a loops while, declarações de caso (switch). No final dos anos 1980, Algol, Pascal e Ada começaram a dar lugar à orientação orientada a objetos linguagens como Smalltalk, C ++ e Eiffel. E assim por diante. Algumas linguagens de programação são projetadas para fins específicos. A linguagem C é boa para programação em nível do sistema (mais próximo ao hardware). Prolog é bom para raciocinar sobre relacionamentos lógicos entre dados. Cada um pode ser usado com sucesso para uma ampla gama de tarefas, mas a ênfase está claramente na especialidade [7].

A linguagem C também tem suas desvantagens em seu uso, por exemplo, ela é extremamente permissiva sobre o que é um programa legal. Essa flexibilidade pode ser boa para profissionais, mas para iniciantes significa apenas que erros de digitação tendem a causar comportamento misterioso, em vez de sinalizar erros, muitas vezes alunos ficam perplexos por horas, porque eles acidentalmente usaram uma vírgula em algum lugar, em vez de um ponto e vírgula, e ao não mostrar o erro ao aluno, ele não poderá consertá-lo, isso acaba atrapalhando a aprendizagem de alunos iniciantes.

Por isso neste trabalho, será abordado as vantagens da linguagem Rust no ensino de programação introdutória.

2.4 A linguagem de programação Rust

Rust é uma linguagem de programação compilada multi-paradigma de propósito geral desenvolvida pela Mozilla pesquisa. Foi publicado inicialmente em 2010 com um primeira versão estável que apareceu em maio de 2015, isto é visando especificamente o domínio atualmente dominado por C e C ++, nomeadamente programação de sistemas, mas diferencia-se por ter sido desenvolvido para prevenir alguns dos problemas relacionados com acessos de memória inválidos (que geram falhas de segmentação) [8].

Historicamente, as linguagens de programação viveram uma dicotomia: achava-se que para ter uma linguagem que fosse segura dever-se-ia desistir da capacidade dela interagir com o hardware em mais baixo nível (ter controle) e com isso perder-se-ia em desempenho também. Do contrário, ter uma linguagem que interaja com o hardware em baixo nível (ter controle), apesar do desempenho obtido com isso, tornaria a mesma menos segura. C ++ cai nessa última categoria. Enquanto o C ++ moderno é significativamente mais seguro do que costumava ser, existem aspectos fundamentais do C ++ que o tornam impossível para estar verdadeiramente seguro.

Rust tenta dar a você uma linguagem com cem por cento de controle, mas de forma absolutamente segura. Ele faz isso por meio de extensa verificação em tempo de compilação, de modo que você paga pouco e geralmente nenhum custo em tempo de execução por seus recursos de segurança. Este problema de segurança não é apenas sobre a conveniência do programador, é uma necessidade que as novas linguagens de programação devem resolver. C e C ++ são inseguros de maneira que causa vulnerabilidades de segurança graves [8].

Um exemplo disso é que a linguagem Rust está sendo usada na plataforma do Android, Sistema Operacional da maioria dos dispositivos móveis atuais. Em uma postagem no site https://security.googleblog.com/, foi dito por Jeff Vander Stoep e Stephen Hines, que fazem parte do Android Team que "além dos esforços contínuos e futuros para melhorar a detecção de bugs de memória, estamos intensificando os esforços para evitá-los. Linguagens seguras para memória são os meios mais econômicos para prevenir bugs de memória. Além de linguagens de memória segura, como Kotlin e Java, temos o prazer de anunciar que o Android Open Source Project (AOSP) agora oferece suporte à linguagem de programação Rust para desenvolver o próprio sistema operacional."

2.5 Metodologia

A Metodologia usada neste trabalho tem os seguintes passos:

- Identificação do problema
- revisão sistemática da literatura
- proposição de solução para resolver o problema
- usar o Rust para ver a sua adequabilidade no ensinos de uma primeira linguagem de programação
- Comparação do uso do C e Rust
- Conclusão

Capítulo 3

Revisão Sistemática da Literatura

Este capítulo irá apresentar uma revisão sistemática da literatura sobre o seguinte assunto: o uso da linguagem de programação Rust nas matérias de programação do curso de Ciência da Computação. Os dados favoráveis, desfavoráveis e inconclusivos sobre o assunto serão apresentados, tendo como objetivo responder a pergunta: Quais os pontos positivos da utilização da linguagem de programação Rust em um curso introdutório de programação?

Para a elaboração da revisão sistemática da literatura foi aplicado a metologia de Chitu Okoli, que conta com oito etapas para sua execução.

Segundo Chitu Okoli [9] revisões da literatura de pesquisa são realizadas para vários propósitos. Eles incluem fornecer uma base teórica para pesquisas subsequentes; aprender a amplitude da pesquisa em um tópico de interesse; ou responder a perguntas práticas, entendendo o que a pesquisa existente tem a dizer sobre o assunto.

3.1 Oito etapas para conduzir uma revisão sistemática da literatura

- Objetivo claro: Requer que o revisor identificar claramente o objetivo e as metas pretendidas da revisão.
- Protocolo e treinamento: para qualquer revisão que emprega mais de um revisor, é essencial que os revisores sejam completamente claros e concordem com o procedimento detalhado a ser seguido. Isso requer um documento detalhado do protocolo e treinamento para todos revisores para garantir consistência na execução da revisão.
- Pesquisando a literatura: O revisor precisa ser explícito ao descrever os detalhes de pesquisa bibliográfica e precisa explicar e justificar como a abrangência do pesquisa foi assegurada.

- Tela prática: também conhecida como triagem para inclusão, esta etapa requer que o revisor seja explícito sobre quais estudos foram considerados para revisão e quais foram eliminado sem exame adicional (uma parte muito necessária de qualquer revisão da literatura).
- Avaliação da qualidade: também conhecido como triagem para exclusão, o revisor precisa explicitamente os critérios para julgar quais artigos têm qualidade insuficiente para serem incluído na síntese da revisão. Todos os artigos incluídos precisam ser pontuados por sua qualidade,dependendo das metodologias de pesquisa empregadas pelos artigos.
- Extração de dados: Após todos os estudos que devem ser incluídos na revisão terem sido identificados, os revisores precisam extrair sistematicamente as informações aplicáveis.
- Síntese de estudos: Também conhecida como análise, esta etapa envolve a combinação dos fatos extraídos dos estudos usando técnicas apropriadas, sejam quantitativas, qualitativas,ou ambos.
- Redigindo a revisão: Além dos princípios padrão a serem seguidos, o processo de revisão sistemática da literatura precisa ser relatado detalhes suficientes para que os resultados da revisão possam ser reproduzidos independentemente.

Para a revisão sistemática da literatura feita neste capítulo algumas etapas foram unificadas com outras etapas que são classificadas separadamente na metodologia de Chitu Okoli, ficando assim dividida em quatro etapas, sendo elas:

- Primeira etapa: Objetivo
- Segunda etapa: Protocolo, Pesquisa e Tela Prática
- Terceira etapa: Avaliação da Qualidade e Síntese de Dados
- Quarta etapa: Extração de dados

Na 3.1 vemos um diagrama das etapas da Revisão Sistemática da Literatura.

3.2 Primeira etapa: Objetivo

Segundo a metodologia de Chitu Okoli [9], numa revisão sistemática da literatura é necessário ter um objetivo claro e metas pretendidas. Portanto o objetivo principal desta revisão sistemática da literatura é reunir informações com a pesquisa realizada para se ter uma conclusão sobre a questão em destaque citada acima.

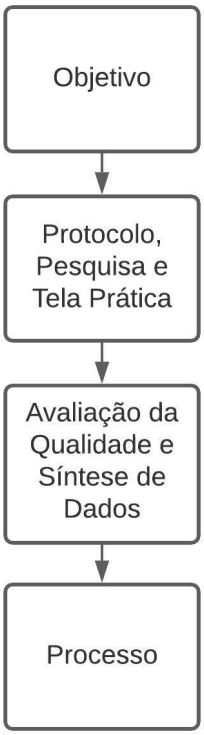

Figura 3.1: Diagrama das etapas da Revisão Sistemática da Literatura

3.3 Segunda etapa: Protocolo, Pesquisa e Tela Prática

Observando os passos apresentado na metodologia de Chitu Okoli [9], o protocolo, a pesquisa e a tela prática foram colocadas na mesma etapa, em que o protocolo seria a escolha da string, um documento contendo todas as informações sobre os artigos selecionados, a pesquisa seria a leitura dos títulos e resumos dos artigos pesquisados, a tela prática seria a seleção dos artigos pesquisados nos sites de pesquisa `https://scholar.google.com.br/`, `https://www.periodicos.capes.gov.br/` e `https://www.google.com/`.

Nessa etapa houveram algumas dificuldades, uma delas seria a falta de artigos científicos a respeito do uso da linguagem de programação Rust no ensino de programação. A causa principal seria a sua criação ser em 2010, com uma primeira versão estável que apareceu em maio de 2015, o que faz com que ela seja uma linguagem de programação nova quando comparamos por exemplo com a linguagem de programação C, que foi criada em 1972.

A segunda dificuldade foi selecionar a string de pesquisa, então foi feita uma pesquisa simples, apresentando a seguinte string - allintitle: "Rust programming"(este comando

procura somente nos títulos dos artigos a string escolhida) no site `https://scholar.google.com.br/` e foram encontrados 42 artigos sobre Rust programming. Tal pesquisa foi feita para pesquisar informações sobre o uso de ensino usando Rust. Dos 42 artigos, um dos artigos foi separado pois havia pesquisas e dados que são relevantes para a pesquisa, o nome do texto é "A cursory overview of the Rust programming language" [8].

Após isso foi feita outra pesquisa no site `https://scholar.google.com.br/`, mas agora com a seguinte string: Rust teaching programming, separando a palavra rust e teaching programming com aspas(o uso das aspas serve para afunilar os proventos e especificar o que é procurado). Foram encontrados 145 artigos, dos quais foram separados 4 artigos, pois eles possuem dados e informações relevantes para a revisão sistemática da literatura.

Tais artigos são: "A functional paradigm using the C language for teaching Programming for Engineers" [10], "A Physical Platform For Teaching Distributed Systems In A Simplified Setting" [11], "Learning and Teaching Algorithm Design and Optimisation Using Contests Tasks" [12]. Também foi feita a pesquisa no site `https://www.google.com/` com a string rust programação paralela Rust e foi encontrado uma monografia com o título "Um panorama sobre Rust" [13], que é usada no curso de Introdução à Computação Paralela e Distribuída na USP pelo professor Alfredo Goldman Vel Lejbman. Por fim foi feita a pesquisa no site `https://www.periodicos.capes.gov.br/` com duas strings: Rust programming language e teaching, foi escolhido dessa forma pois o site tem a opção de pesquisar duas strings que necessariamente devem conter as ambas no artigo, e como a primeira string são 3 palavras, necessariamente devem conte-las, pois Rust é um termo usado na língua inglesa e influenciaria na quantidade de artigos. Desta pesquisa foi encontrado um texto que possui informações relevantes para a revisão sistemática da literatura, o artigo "From Theory to Systems: A Grounded Approach to Programming Language Education" [14].

O critério de exclusão dos artigos eram artigos que não se tratavam sobre ensino de programação ou sobre a linguagem Rust.

Pela falta de artigos de pesquisa em uma sala de aula e também sobre a linguagem de programação Rust, alguns dos artigos selecionados apresentam pesquisas sobre outras linguagens de programação, como por exemplo C, mas apresentam pontos a favor do uso de Rust no ensino de programação ou apresentam formas de ensino que poderiam ser aplicadas no ensino de programação usando Rust.

Na tabela Tabela 3.1 abaixo foi feita a indexação dos artigos selecionados.

Tabela 3.1: Indexação dos artigos.

Nome do artigo	ID
A cursory overview of the Rust programming language	A1
A Functional Paradigm using the C Language for Teaching Programming for Engineers	A2
A Physical Platform For Teaching Distributed Systems In A Simplified Setting	A3
From Theory to Systems: A Grounded Approach to Programming Language Education	A4
Monografia na matéria MAC 5742: Introdução à Computação Paralela e Distribuída Professor Alfredo Goldman Vel Lejbman IME USP	A5

3.4 Terceira etapa: Avaliação da Qualidade e Síntese de Dados

Seguindo os passos apresentado na metodologia de Chitu Okoli [9], a etapa de avaliação dos artigos foi combinada com a etapa de síntese de dados, assim foi usado um método de avaliação da qualidade que contém três critérios que são pontuados de 1 a 3, onde 1 representa baixa, 2 representa média e 3 representa alta pontuação e o total seria o somatório das pontuações dos três critérios.

Tal método foi baseado, com algumas adaptações, na Revisão Sistemática da literatura "Gamification for health and wellbeing: A systematic review of the literature" [15].

Os critérios utilizados foram:

- **C1**: O tema principal do artigo é sobre a linguagem de programação Rust?
- **C2**: Qual o tipo de pesquisa foi utilizada no artigo?
- **C3**: Quão apropriados são os métodos e análises?

Para avaliar os artigos em relação ao critério C1 foi verificado se o tema principal do artigo é sobre a linguagem de programação Rust, sendo (3) a resposta sim, (2) a resposta não, mas apresentam mais de três pontos ou mais sobre Rust e (1) a resposta não, mas apresenta um ou dois pontos sobre Rust. Para avaliar os artigos em relação ao critério C2, foi verificado os tipos de pesquisa utilizada nos artigos, sendo (3) a estudos que apresentam um grupo-controle, ou seja, aplicação em sala de aula, (2) a estudos feitos através de dados mais apurados, como por exemplo uma tabela, um gráfico e (1) a estudos feitos somente com referência bibliográfica de autores.

Para avaliar C3, foi atribuída nota um (1) para cada técnica de coleta ou análise de dados, como por exemplo: entrevistas, questionários, observação direta. Limitando-se até a nota três.

Tabela 3.2: Critérios de Avaliação.

Artigo(ID)	C1	C2	C3	Total
A1	3	2	2	7
A2	1	3	3	6
A3	3	2	2	7
A4	3	3	3	9
A5	3	3	3	9
Média	2,6	2,6	2,6	7,6

Assim foi feita a Tabela 3.2.

3.5 Quarta etapa: Extração de dados

Essa etapa consiste na extração sistemática de dados relevantes dos artigos selecionados, como por exemplo, qual tipo de pesquisa foi feita. No artigo "A cursory overview of the Rust programming language" [8] é destacado vários pontos positivos a respeito do Rust, como por exemplo, mover semântica, segurança de memória garantida, threads sem corridas de dados, correspondência de padrões, inferência de tipo, tempo de execução mínimo, ligações C mais eficientes, genéricos baseados em características, permitem implementações no estilo Venndiagram. Resumindo melhor cada recurso apresentado acima, o próprio site da linguagem de programação Rust define que:

> Rust é uma linguagem de programação de sistemas que funciona incrivelmente rápido, evita falhas de segurança, e garante a segurança da linha. `https://prev.rust-lang.org/pt-BR/`

No artigo A1 também é apresentado as especificações que são exclusivas do Rust, como por exemplo, o sistema de *ownership*, que traduzindo para o português seria propriedade, e esse significado está diretamente relacionado a sua forma de ser usada, pois as variáveis possuem os recursos de dados alocados (como vetores, pilhas e outros objetos) aos quais estão vinculados e isso faz com que cada vinculação de variável deva possuir exatamente um recurso de dados.

Outra especificação seria o gerenciamento manual de memória e isso faz com que o programador esteja no controle completo de qual memória está sendo alocada, assim como em C.

Porém ao contrário da linguagem de programação C, os programadores podem não estar cientes de quanto controle eles têm, enquanto no Rust eles sabe automaticamente quando alocar e liberar memória. Com isso Rust parece uma linguagem de alto nível, mas pode ser usada como uma linguagem de baixo nível também.

Outros conceitos específicos da linguagem de programação Rust seria o conceito *type-system*[1], em que é complexo (mas não complicado), pois não afeta o usuário final além de garantias em tempo de compilação. O *type-system* define três áreas nos quais os objetos podem ser alocados: a pilha, a pilha local e a troca pilha. Essas áreas têm um ponteiro correspondente tipos: o ponteiro emprestado, a caixa compartilhada, e a caixa exclusiva. Caixas são chamadas de *affine type*[2]. Isso significa que o compilador Rust, em tempo de compilação, determina quando a caixa entra e sai do escopo, e insere as chamadas apropriadas lá.

Por fim o artigo A1 aborda que o ponto de interrogação maior, é quão bem Rust executa contra outras línguagens de programação que também visam ser rápidas e seguras. Ter o melhor desempenho seria uma grande vantagem para ganhar popularidade entre os programadores, e isso poderia muito bem ser o caso, considerando o quão perto Rust já está de C/C ++ neste aspecto.

Portanto na extração de dados artigo A1 foi destacado os pontos positivos da linguagem de programação Rust, focando principalmente em suas especificações exclusivas.

Prosseguindo na extração de dados para a revisão sistemática destaca-se no artigo "A functional paradigm using the C language for teaching Programming for Engineers" [10] a metodologia usada para o ensino de programação na Universidade Simón Bolívar em Caracas, que consiste em fortalecer e melhorar a experiência de ensino e aprendizagem, incorporando o seguinte método híbrido, que empresta conceitos e estruturas da programação funcional de paradigma, e é implementado seletivamente usando macro adequada na linguagem C associada com extensões de compilador. Assim os seguintes pontos foram implementados na metodologia:

- Abordagem funcional desde o início, mais orientada para declarar mudanças do que para atribuição de variável;
- Especificações formais simplificadas permitindo a verificação de pré e pós-condições, invariantes e limites de índice sobre expressões algorítmicas.
- Introdução inicial de recursão para soluções algorítmicas, com ênfase particular na recursão de cauda.

[1] *Type-system* funciona como uma ferramenta formal para fazer análises matemáticas da linguagem; por outro lado, é um método formal para projetar e implementar uma linguagem de maneira rigorosa e precisa. [16]

[2] *Affine type*: quando há mais de um uso ou pelo menos um uso de uma função, expressão ou estrutura de dados é necessário habilitar otimizações nos compiladores. Se houver no máximo um uso de um objeto, então dizemos que o objeto tem *affine type*. [17]

- Tipos abstratos de dados (ADTs) implementados como tipos de ponteiro de estrutura em uma programação de aplicativo de interface (API), armazenada em bibliotecas de código modulares.
- Abstrações de funções de ordem superior para implementar ADT's tradicionais genéricos (coleção, conjunto, empilhamento, fila, sequência, listas com link simples e duplo).
- Usando um Coletor de Lixo para gerenciamento inteligente de alocação de memória dinâmica.
- Uso de um IDE com depurador integrado e dinâmico, e ferramentas de validação de memória.

Essa abordagem foi executada três vezes consecutivas nos períodos, e embora seja evidentemente claro que não há dados suficientes no período de tempo para fazer uma afirmação apoiada por estatísticas sólidas, o número dos temas abordados nos cursos aumentaram em 20 por cento, em relação aos períodos que apresentavam a metodologia regular de ensino de programação usando a linguagem C. E permitiu cobrir alguns deles com maior profundidade. os problemas e projetos de laboratório aumentaram em complexidade, e o mesmo aconteceu com as soluções propostas, destacando uma maior compreensão por parte dos alunos do processo algorítmico funcional, apropriado para disciplinas de engenharia que já se baseiam na agregação de componentes funcionais. Os resultados revelam um salto notável na qualidade dos trabalhos de casa e tarefas de codificação de laboratório e legibilidade incrementada. E no artigo concluiu-se que seria mais fácil, simplesmente usar uma linguagem com um verdadeiro paradigma funcional, como por exemplo o Rust, para aplicar a metodologia apresentada. Sendo assim este é um dado importante que foi extraído do artigo, pois seria possível a aplicação da metologia de ensino usando programação funcional, utilizando-se a linguagem de programação Rust.

Prosseguindo na extração de dados, no artigo "A Physical Platform For Teaching Distributed Systems In A Simplified Setting" [11] que tem como objetivo a implementação de uma plataforma para ensinar o básico da programação bare metal[3] para os alunos e a avaliação da utilidade pedagógica da plataforma e como implementar a ideia para motivar os alunos. A plataforma consiste em uma placa-mãe, com nós programáveis e nós periféricos. Os nós são programados usando um interface de usuário baseada na web fácil de usar, destinada a ser veiculada em um servidor em execução na sala de aula. E foi usado Rust para programar os Node MCUs. Isso foi porque o Rust tem recursos de segurança mais aprimorados em relação ao C (como ponteiros com vida útil, semântica de

[3]A programação bare-metal interage com um sistema no nível do hardware, levando em consideração a construção específica do hardware.

movimento e coletor de lixo), bem como o sistema de tipos aprimorado. O protótipo foi projetado para ser muito simples e com pouca configuração do usuário final necessária para motivar os alunos. O objetivo principal é que seja fácil começar, mas com a oportunidade de desenvolver tarefas desafiadoras para alunos mais avançados para praticar. Portanto o PicoNodes pode ajudar os professores a cumprir os seguintes objetivos na disciplina de Tecnologia(para as séries 7 a 9):

- Alunos são forçados a considerar como seus componentes devem interagir entre si.
- Discussão de soluções técnicas e várias tarefas com muitas possibilidades soluções, com vários prós e contras, que os alunos irão deseja, comparar e contrastar.
- Os prós e contras dos sistemas distribuídos, os alunos que já usaram outros sistemas (como o Scratch) provavelmente notarão que o sistema distribuído é geralmente mais difícil de raciocinar
- A depuração, que é um bom ponto de partida para a discussão sobre quando esses sistemas são usados na prática.

Nota-se dos dados extraídos que a plataforma foi criada para o ensino de pensamento computacional para alunos da 7 a 9 série, porém ela pode ser adaptada para o ensino de programação, além disso a plataforma é criada com a linguagem de programação Rust, isso faz com que ela possa ser até um projeto criado pelos próprios alunos de matérias mais avançadas de programação, como sistemas operacionais, estrutura de dados, além de ter a oportunidade de ensinar a linguagem Rust e verificar seus prós e contras como linguagem de programação, visto que é necessário para o projeto uma linguagem de programação que cuide da parte do hardware, um dos pontos fortes do Rust.

Prosseguindo a extração de dados, destaca-se no artigo "From Theory to Systems: A Grounded Approach to Programming Language Education" [14] uma nova abordagem para o ensino de um curso de linguagens de programação de pós-graduação com foco em usar ideias de programação de sistemas e linguagens como WebAssembly e Rust para motivar a teoria de linguagem e programação. Com base na experiência anterior dos alunos com idiomas de baixo nível, o curso mostra como os sistemas de tipos e a teoria PL são usadas para evitar erros complicados do mundo real que os alunos encontram na prática. Portando refletindo sobre o projeto curricular e as lições aprendidas em dois anos de ensino de programação em Stanford, percebe-se que a integração de ideias de sistemas pode fornecer aos alunos uma educação mais fundamentada e agradável em linguagens de programação. Para tal, acredita-se que o principal benefício de aprender a teoria da linguagem de programação é fornecer um modelo simples de conceitos fundamentais em computação, ou seja, não apenas saber como programar, mas compreender a essência

das ideias como variáveis, funções e tipos. Assim o curso tem como ementa as seguintes divisões: Lógica e semântica, teoria dos tipos, programação funcional, WebAssembly, Rust, tipos de sessão, tipagem dinâmica e projeto final. Esta versão do currículo foi ensinada no outono de 2018 para uma classe de 77 alunos de Stanford, principalmente pós-graduados de ciência da computação, e principalmente mestres e alunos de graduação de nível superior. 85% dos alunos forneceram feedback anônimo sobre o cursos. Alunos foram questionados "Quanto você aprendeu com este curso?"com respostas de (5)"a maioria", (4)“Muito”, (3)“Uma quantidade moderada”, (2)“Um pouco”, (1)“Nada”.A média de resposta foi de 4,3 com 87% dos alunos respondendo 4 ou mais.

Quando solicitados a articular habilidades aprendidas, a maioria dos entrevistados escreveu sobre: 1) Teoria de PL e provas sobre programas, 2) programação funcional e novos modelos de computação, ou 3) aprender a adquirir rapidamente novos idiomas. Eu acredito que esta variedade de respostas reflete positivamente no currículo, pois diferentes alunos entram com diferentes objetivos de aprendizagem pessoal e a amplitude do curso permite que os alunos encontrem o aspecto das linguagens de programação de que mais gostam.

Com base nos dados extraídos este foi o texto de mais relevância da pesquisa feita, pois apresenta uma metodologia para o ensino de programação computacional utilizando-se a linguagem Rust(boa parte do curso) e programação funcional, respondendo assim a pergunta feita no início da revisão sistemática da literatura, Rust pode ser usada no ensino de matérias de programação?

Prosseguindo na extração de dados dos artigos, será extraído a metodologia usada no curso Introdução à Computação Paralela e Distribuída da USP, que produziu uma monografia com o título: "Um panorama sobre Rust" [13].

Foi usado no curso ferramentas incluídas na distribuição de Rust:

- Rustc: O compilador de Rust é um frontend do llvm e por isso tem as opções de backend do llvm. O próprio compilador compila ele mesmo desde 2011.
- Rustdoc: Uma ferramenta para gerar documentação a partir do código fonte que pode conter descrições em MarkDown que ajuda na consistência da documentação em relação ao código fonte.
- Rust-gdb: Uma versão do GNU debugger para Rust.
- Cargo: O Cargo é um gerenciador de projetos para Rust. A ferramenta usa rustdoc para a geração de documentação, executa testes e benchmarks.
- IDE: Apesar de ser um projeto bastante novo já existem vários plugins para as ferramentas comuns que apoiam o desenvolvimento em Rust. Além disso há uma versão do ambiente de trabalho Eclipse: RustDT [rusd].

Os conceitos de ownership, programação paralela, threads, canais e memória compartilhada são utilizados no curso. E o curso foi repetido em três semestres, segundo de 2016, primeiro de 2017 e o primeiro de 2018. Porém pelo fato de Rust ser uma linguagem muito nova ainda faltam algumas caraterísticas nas bibliotecas básicas sobretudo na área de paralelização. Supomos que essa falta será suprida brevemente porque já existe uma comunidade significativa bem como uma empresa claramente interessada no desenvolvimento de Rust.

Essa monografia foi escolhida por ser um curso de programação paralela e distribuída na USP e isso responde diretamente a pergunta trazida nessa revisão sistemática: Quais os pontos positivos da utilização da linguagem de programação Rust em um curso introdutório de programação?

Capítulo 4

Comparação do Rust com C

Neste capítulo será feito a comparação da linguagem de programação Rust com C. Visto que a linguagem de programação C é usada na disciplina Algoritmos e Programação de Computadores(APC) da Universidade de Brasília (UnB) com os seguintes conteúdos na sua ementa:

- Especificação de variáveis, Identificadores, valores e tipos.
- Operadores e expressões.
- Estruturas de controle condicional.
- Estruturas de repetição.
- Funções.
- Entrada e saída de dados.
- Estruturas de dados estáticas: agregados homogêneos e heterogêneos. (vetores, matrizes, strings e registros)
- Ponteiros.

Portanto a comparação será feita em cada conteúdo mencionado acima, e será feito também observações quanto a vantagens e desvantagens de cada linguagem no aprendizado de programação em cada item acima.

4.1 Comparação de C e Rust nas IDEs utilizadas

Primeiramente veremos quais tipos de IDE ou ambiente de desenvolvimento integrado são utilizados para o desenvolvimentos dos programas. No curso de APC presencial era utilizado o codeblocks, ele é um IDE C, C ++ e Fortran gratuito projetado para ser

```c
#include <stdio.h>
main(void){
    int n = 1;
    float saldo = 1000;
    printf("O numero total
    de contas e:%d\n",n);
    saldo = saldo + 1000;
    printf("Saldo: %f\n",saldo);
    saldo = saldo - 1000;
    printf("Saldo: %f\n",saldo);
    saldo = saldo * 2;
    printf("Saldo: %f\n",saldo);
    saldo = saldo/2;
    printf("Saldo: %f\n",saldo);
    return 0;
}
```

```rust
fn main() {
    let n: i32 = 1;
    let saldo: f64 = 1000.0;
    println!("O numero total de
    contas e:{}",n);
    println!("Saldo: {}",saldo);
    let saldo = saldo + 1000.0;
    println!("Saldo: {}",saldo);
    let saldo = saldo - 1000.0;
    println!("Saldo: {}",saldo);
    let saldo = saldo * 2.0;
    println!("Saldo: {}",saldo);
    let saldo = saldo / 2.0;
    println!("Saldo: {}",saldo);

}
```

Figura 4.1: Comparação de código em C e Rust na definição de variáveis, uso de expressões e operadores

muito extensível e totalmente configurável, de acordo com o próprio site do codeblocks `http://www.codeblocks.org/`.

Já a linguagem de programação Rust possui um ambiente de desenvolvimento, em que todas as ferramentas são instaladas inicialmente, e o conjunto de ferramentas são rustc, cargo e rustup.

Com isso o Rust possui uma ótima documentação, um compilador amigável com mensagens de erros úteis, e ferramental de primeira qualidade integrada de compilação e gerenciamento de pacotes, suporte inteligente para múltiplos editores com autocompleção e inspeções de tipos, um formatador automático. Uma das ferramentas que chama atenção é o Cargo, que não oferece muito valor em comparação com apenas o uso do rustc, mas vai provar seu valor à medida que seus programas se tornam mais complexos. Com projetos complexos compostos de várias caixas, é muito mais fácil deixar Cargo coordenar a construção, de acordo com o próprio site do Rust, `https://www.rust-lang.org/`.

Portanto é observado que a linguagem Rust possui várias ferramentas que agregam no ensino da programação, enquanto o C precisa de um IDE para facilitar a elaboração de programas, o Rust possui um ambiente de desenvolvimento que é instalado em conjunto com sua instalação principal.

4.2 Comparação de C e Rust na definição de variáveis, uso de expressões e operadores

O código em C que defini variáveis, uso de expressão e operadores, está exposto no repositório do github no seguinte endereço:

`https://github.com/DaniloFukuda/TCC-ICC-DaniloFukuda/blob/Codigos-em-C/C1.c`.

O código em Rust, está exposto no repositório do gitbuh no seguinte endereço:

`https://github.com/DaniloFukuda/TCC-ICC-DaniloFukuda/tree/Codigo-em-Rust/programa1`

Normalmente o aprendizado de uma linguagem de programação se inicia pela definição de tipos, variáveis, operações e expressões. Para iniciar a comparação entre as linguagens de programação C e Rust será feito um programa simulando operações bancárias usando as duas linguagens. Primeiramente serão criados variáveis de dois tipos e serão implementadas expressões com operações entre as mesmas, conforme na Figura 4.1.

Comparando-se as duas linguagens apenas em relação aos tipos, variáveis e expressões percebe-se apenas uma pequena diferença na sintaxe.

Em relação à semântica das operações usa-se o comando let na linguagem Rust. Ela é utilizada para definir variáveis em geral, como por exemplo: let x:i32 = 0, essa expressão cria uma variável inteira x com 32 bits e que tem como valor o numero 0, assim necessariamente para a definição de variáveis é necessário o comando let.

Outra diferença sintática seria o uso dos dois pontos para definir o tipo da variável, tendo em vista de que pode ser criado uma variável de 32 bit ou de 16 bits, oferecendo mais opções de uso para os programadores.

Enquanto na linguagem C é somente utilizado a definição de variáveis colocando o tipo, nome e valor da variável, conforme mostra na linha 3 da Figura 4.1.

Comparando o tratamento dos erros em ambas linguagens, o Rust tem pontos positivos para o uso nas matérias introdutórias de programação.

No site oficial da linguagem Rust, é disponibilizado a qualquer pessoa um curso introdutório para ensinar sua sintaxe, no endereço a seguir:

`https://doc.rust-lang.org/book/ch00-00-introduction.html`.

Neste link é destacado que a linguagem Rust traz ferramentas de desenvolvedor contemporâneas para o mundo da programação de sistemas:

- Cargo, o gerenciador de dependências incluído e ferramenta de construção, torna a adição, compilação e gerenciamento de dependências indolor e consistente em todo o ecossistema Rust.

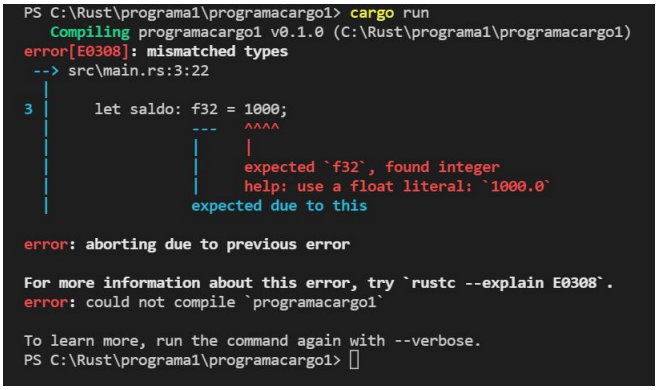

Figura 4.2: Exemplo 1 da interface dos erros

- Rustfmt garante um estilo de codificação consistente entre os desenvolvedores.
- O Rust Language Server capacita a integração IDE (Integrated Development Environment) para conclusão de código e mensagens de erro em linha.

Ao usar essas e outras ferramentas no ecossistema Rust, os desenvolvedores podem ser produtivos ao escrever código em nível de sistema. No código 1 em Rust da Figura 4.1, foi visto que ao haver erros no código, comentários e sugestões aparecem para auxiliar o programador, por exemplo, na linha 3, se o programador definir a variável saldo dessa forma: "let saldo: f32 = 1000;", aparace logo abaixo o nome do erro:"mismatched types", e ainda um comentário:"expected ‘f32‘, found integer","help: use a float literal: ‘1000.0‘". Assim para corrigir o erro é necessário somente mudar o código para "let saldo: f32 = 1000.0;". Pois ao declarar a variavel saldo como float, é esperado que o valor seja float também, assim é necessário colocar o ".0"depois do valor 1000. Tudo isso é mostrado na Figura 4.2

4.3 Comparação de C e Rust em Estruturas de controle de condição

O código 2 em C, como visto na Figura 4.3 utiliza-se de estruturas de controle, switch-case e if-else, e simula um menu simples para fazer transações bancárias. E está exposto no repositório do github no seguinte endereço: `https://github.com/DaniloFukuda/TCC-ICC-DaniloFukuda/blob/Codigos-em-C/c2.c`.

O código 2 em Rust, está exposto no repositório do github no seguinte endereço: `https://github.com/DaniloFukuda/TCC-ICC-DaniloFukuda/tree/Codigo-em-Rust/programa2`.

No código em Rust, como visto na imagem 4.3, foi feito o mesmo menu, porém na linguagem de programação Rust, não existe a estrutura de controle switch-case, porém existe

```c
#include <stdio.h>
main(void){
int num;
float saldo = 1000, dep = 0, saque = 0;

printf("Selecione a operacao:\n");
printf("1 = Deposito\n");
printf("2 = Saque\n");
scanf("%d",&num);
switch(num){
    case 1:
        printf("Qual a quantia a ser depositada?\n");
        scanf("%f",&dep);
        saldo = saldo + dep;
        printf("O saldo e de: %f",saldo);
        break;
    case 2:
        printf("Qual a quantia a ser sacada?\n");
        scanf("%f",&saque);
        if(saque <= saldo){
            saldo = saldo - saque;
            printf("O saldo e de: %f\n",saldo);
            break;
        }
        else
            printf("Valor invalido, saque > saldo\n");
            break;
    default :
        printf ("Valor invalido!\n");

}
return 0;
}
```

```rust
use std::io;
fn main() {
let mut saldo: f32 = 1000.0;
println!("Selecione a operacao");
println!("1 = Deposito");
println!("2 = Saque");
let mut num = String::new();
io::stdin().read_line(&mut num)
        .expect("Falha ao ler entrada");
let num2:i32= num.trim().parse().expect("invalid input");
if num2 == 1{
    println!("Qual a quantia a ser depositada");
    let mut dep = String::new();
    io::stdin().read_line(&mut dep)
        .expect("Falha ao ler entrada");
    let dep2:f32= dep.trim().parse().expect("invalid input");
    saldo = saldo + dep2;
    println!("O saldo e de: {}",saldo);
    }else if num2 == 2 {
        println!("Qual a quantia a ser sacada");
        let mut saque = String::new();
        io::stdin().read_line(&mut saque)
            .expect("Falha ao ler entrada");
        let saque2:f32= saque.trim().parse().expect("invalid input");
        if saldo >= saque2{
            saldo = saldo - saque2;
            println!("O saldo e de: {}",saldo);
        }else{
            println!("Valor invalido, saque > saldo");
        }
    }else{
        println! ("Valor invalido");
    }
}
```

Figura 4.3: Comparação de código C e Rust em Estruturas de controle condicional

uma estrutura parecida, que é a função match. Mas optou-se por não usar tal função, usando em seu lugar if e else que facilitaria o entendimento do código para programadores iniciantes.

Comparando os dois códigos percebe-se o código da linguagem Rust ficou maior, isso porque o Rust tem mais rigor ao lidar com os tipos de dados. Por exemplo nas linhas 7,8 e 9 foram feitos comandos para receber uma entrada dada pelo usuário, e na linha 10 do código, foi transformado a string que recebeu a entrada do teclado, em uma variável de número inteiro e nas linhas 16 e 24 foi transformado em uma variável de tipo float. Vários comandos foram usados nessas linhas tanto para ler entradas do teclado e tanto para transformar uma string em outros tipos de variável, como por exemplo, float de 32 bits e inteiros de 32 bits. Um desses comandos seria o "readline"que leem entradas do teclado e o "String::new()"que cria uma variável String. Já os comando "trim()"e "parse()"servem para transformam o conteúdo da variável String para uma variável inteira ou float de 32 bits.

Outra diferença na sintaxe é o uso do comando mut após let, fazendo assim a variável se tornar mutável, um exemplo simples seria colocar a palavra mut no exemplo dado acima: let mut x:i32 = 0. isso faz com que a variável x se torne mutável.

Observando também no código em Rust, percebe-se que foi necessário utilizar uma biblioteca extra nesse código, a "std". Ela foi utilizada para ler as entradas do teclado para interagir com o usuário no menu bancário.

Já na linguagem C é utilizado a função scanf para ler as entradas do teclado. Outra ponto importante seria a o uso do switch-case, que sua abstração é bem coerente com a estrutura do código 2, que é um menu bancário, dividindo suas opções em 3 e se não for uma dessas opções apresenta na tela que o valor digitado é inválido.

Observando somente a sintaxe das linguagens, ambas possuem uma abstração para ler as entradas do teclado, que é o endereço da variável. O aluno pode ter complicações ao lidar com tal abstração, porém é de extrema importância para sua aprendizagem na programação, sendo um conteúdo primordial para o assunto. Isso também vale para as estruturas de condição, no caso da linguagem C seria o switch-case e no caso da linguagem Rust seria o if-else.

Observando o tratamento dos erros no código 2 da Figura 4.3, a linguagem Rust novamente tem pontos positivos para o uso em turmas introdutórias de programação. Por exemplo, na linha 3, se tirarmos a palavra mut do código: "let saldo: f32 = 1000.0;", o erro é mostrado como: "cannot assign twice to immutable variable ‘saldo‘", e o comentário mostrado é: "first assignment to ‘saldo‘help: make this binding mutable: ‘mut saldo‘". Então para corrigir o erro é necessário colocar palavra mut depois de let, como está no código em Rust da Figura 4.3. O erro é mostrado porque a variável saldo é definida

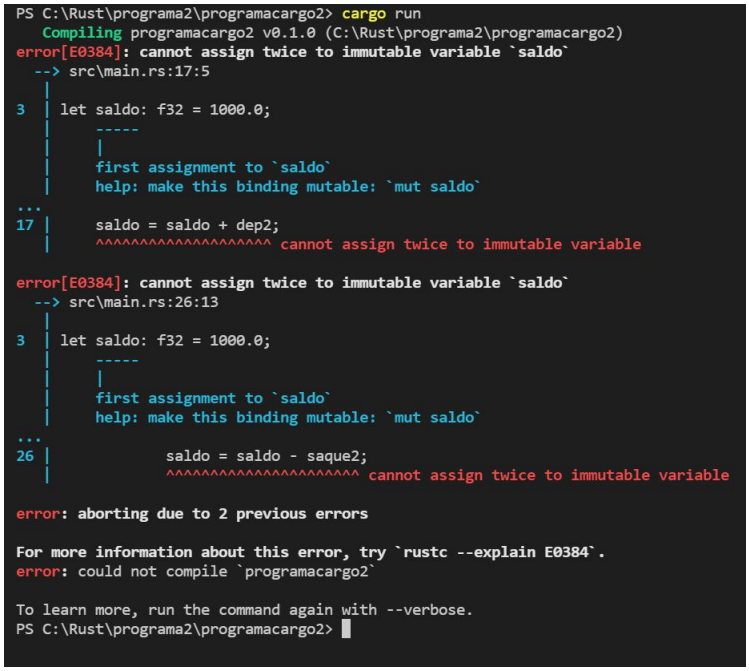

Figura 4.4: Exemplo 2 da interface dos erros

novamente no decorrer do código, na linha 26, sendo necessário declará-la como uma variável mutável. Outra observação sobre isso é que no Rust existe a opção de criar uma variável imutável, dando ao programador mais controle do seu código e assim mais segurança para o mesmo. Tudo isso é apresentado na Figura 4.4.

4.4 Comparação de C e Rust em Estruturas de repetição, Vetores e ponteiros

O código 3 em C que utiliza-se dos conceitos de estruturas de repetição, vetores e ponteiros está exposto no repositório do github no seguinte endereço: `https://github.com/DaniloFukuda/TCC-ICC-DaniloFukuda/blob/Codigos-em-C/c3.c`. No código em C, como visto na Figura 4.5, foi feito um programa para simular transações bancárias. Com opções para se criar, apagar, depositar, sacar e mostrar as contas. A estrutura usada para criar as contas foi um vetor de struct, em que a struct contém os seguintes dados: numero e saldo da conta. Foi utilizado também a função malloc para fazer uma alocação dinâmica do vetor da struct e estruturas de repetição para manter o menu repetindo sempre que terminar uma operação, mas também com a opção de sair do menu, saindo assim também da estrutura de repetição.

O código 3 em Rust está exposto no repositório do github no seguinte endereço:

```
.
#include <stdio.h>
typedef struct{
    int numero;
    float saldo;
}Bconta;
void imprimir(Bconta *conta, int i){
int j = 0;
for(j = 0; j<=i; j++){
    printf("conta: %d\n",conta[j].numero);
    printf("saldo da conta:%f\n",conta[j].saldo);
}
return NULL;
}
int main(void){
int num, x = 0, i = 0, j = 0,k = 0,l = 0,apagar = 0, acesso = 0;
float dep = 0, saque = 0;
Bconta *conta;
conta = (Bconta *) malloc(20 * sizeof(Bconta));
conta[i].numero = i;
conta[i].saldo = 0;
while(x == 0){
    printf("Escolha qual menu quer acessar\n");
    printf("1 = Menu de administrador\n");
    printf("2 = Menu de usuario\n");
    printf("3 = Sair\n");
    scanf("%d",&num);
        switch(num){
            case 1:
                printf("1 - Mostrar contas\n");
                printf("2 - Criar contas\n");
                printf("3 - Apagar contas\n");
                printf("4 - Sair\n");
                scanf("%d",&num);
                switch(num){
                    case 1:
                        imprimir(&conta[0],i);
                        break;
                    case 2:
                        printf("Quantas contas serao criadas?\n");
                        scanf("%d",&k);
                        if(i + k<20){
                            for (j = 0; j <= k; ++j){
                                conta[i].numero = i;
                                conta[i].saldo = 0;
                                printf("saldo da conta %d:%f\n",conta[j].numero,conta[j].saldo);
                                i++;
                            }
                            i--;
                            printf("Valor de i: %d\n",i);
                        }
                        else
                            printf("Valor invalido!\n");
                        break;
                    case 3:
                        printf("Quantas contas vao ser apagadas?");
                        scanf("%d",&apagar);
                        if(i > apagar){
                            i = i - apagar;
                        }
                        else
                            printf("Numero invalido\n");
                        break;
                    case 4:
                        printf ("Saiu com sucesso!\n");
                        x= 1;
                        break;
                    default :
                        printf ("Valor invalido!\n");
                }
        case 2:
                printf("Qual conta sera acessada?");
                scanf("%d",&acesso);
                if(acesso <= i){
                    printf("Selecione a operacao:\n");
                    printf("1 = Deposito\n");
                    printf("2 = Saque\n");
                    printf("3 - Saldo\n");
                    printf("4 = Sair\n");
                    scanf("%d",&num);
                    switch(num){
                        case 1:
                            printf("Qual a quantia a ser depositada?\n");
                            scanf("%f",&dep);
                            conta[acesso].saldo = conta[acesso].saldo + dep;
                            printf("O saldo e de: %f\n",conta[acesso].saldo);
                        break;
                        case 2:
                            printf("Qual a quantia a ser sacada?\n");
                            scanf("%f",&saque);
                            if(saque <= conta[acesso].saldo){
                                conta[acesso].saldo = conta[acesso].saldo - saque;
                                printf("O saldo e de:%f\n",conta[acesso].saldo);
                            }
                            else{
                                printf("Valor invalido, saque > saldo\n");
                            }
                            break;
                        case 3:
                            printf("O saldo da conta %d e de: %f\n",conta[acesso].numero,conta[acesso].saldo);
                            break;
                        case 4:
                            printf ("Saiu com sucesso!\n");
                            x = 1;
                            break;
                        default :
                            printf ("Valor invalido!\n");
                    }
                }
                else
                    printf("Acesso invalido!\n");
                break;
        case 3:
                printf ("Saiu com sucesso!\n");
                x = 1;
                break;
        default :
                printf ("Valor invalido!\n");
    }
}
free(conta);
return 0;
}
```

Figura 4.5: Código C: Estruturas de repetição, vetores e ponteiros

```
.

use std::io;

fn main() {
let mut saida0:i32 = 0;
let mut saida1:i32 = 0;
let mut i: i32 = 0;
let mut j: i32 = 0;
let mut saida2: i32 = 0;
let mut conta:Vec<f64> = Vec::new();
while saida0 == 0{
    println!("Escolha qual menu quer acessar");
    println!("1 = Menu de administrador");
    println!("2 = Menu de usuario");
    println!("3 = Sair");
    let mut num = String::new();
    io::stdin().read_line(&mut num)
        .expect("Falha ao ler entrada");
    let num2:i32= num.trim().parse().expect("invalid input");
    if num2 == 1{
        while saida1 == 0{
            println!("1 - Mostrar contas");
            println!("2 - Criar contas");
            println!("3 - Apagar contas");
            println!("4 - Sair");
            let mut num = String::new();
            io::stdin().read_line(&mut num)
                .expect("Falha ao ler entrada");
            let num2:i32= num.trim().parse().expect("invalid input");
            if num2 == 1{
                if i == 0{
                    print!("Nao existem contas criadas");
                    print!("
");
                }
                else{
                    while j <i{
                        println!("Conta{} : Saldo de {}", j, conta[j as usize]);
                        j = j + 1;
                    }
                    j = 0;
                }
            }
            else if num2 == 2{
                println!("Quantas contas serao criadas?");
                let mut cont = String::new();
                io::stdin().read_line(&mut cont)
                    .expect("Falha ao ler entrada");
                let cont2:i32= cont.trim().parse().expect("invalid input");
                println!("{} serao criadas",cont2);
                while i < cont2{
                conta.push(0.0);
                i = i + 1;
                }
            }
            else if num2 == 3{
                println!("Quantas contas ser o apagadas?");
                let mut _del = String::new();
                io::stdin().read_line(&mut _del)
                    .expect("Falha ao ler entrada");
                let _del2:i32= num.trim().parse().expect("invalid input");
                if i == 0{
                    println!("N o existem contas para apagar!");
                }
                else{
                    i = i - num2;
                }
            }
            else if num2 == 4{
                println!("Voce voltara ao menu anterior!");
                saida1 = 1;
            }
            else{
                println!("Numero invalido!")
            }
        }
    }
    else if num2 == 2{
        println!("Qual conta ser  acessada?");
        let mut acesso1 = String::new();
        io::stdin().read_line(&mut acesso1)
            .expect("Falha ao ler entrada");
        let acesso:i32= acesso1.trim().parse().expect("invalid input");
            if acesso <= i{
                while saida2 == 0{
                    println!("Selecione a operacao");
                    println!("1 = Deposito");
                    println!("2 = Saque");
                    println!("3 = Saldo");
                    println!("4 = Sair");
                    let mut num3 = String::new();
                    io::stdin().read_line(&mut num3)
                        .expect("Falha ao ler entrada");
                    let num4:i32= num3.trim().parse().expect("invalid input");
                    if num4 == 1{
                        println!("Qual a quantia a ser depositada");
                        let mut dep = String::new();
                        io::stdin().read_line(&mut dep)
                            .expect("Falha ao ler entrada");
                        let dep1:f64= dep.trim().parse().expect("invalid input");
                        conta[acesso as usize] = conta[acesso as usize] + dep1;
                        println!("O saldo e de: {}",conta[acesso as usize]);
                    }else if num4 == 2 {
                        println!("Qual a quantia a ser sacada");
                        let mut saque = String::new();
                        io::stdin().read_line(&mut saque)
                            .expect("Falha ao ler entrada");
                        let saque2:f64= saque.trim().parse().expect("invalid input");
                        if conta[acesso as usize] >= saque2{
                            conta[acesso as usize] = conta[acesso as usize] - saque2;
                            println!("O saldo e de: {}",conta[acesso as usize]);
                        }else{
                            println!("Valor invalido, saque > saldo da conta{}",acesso);
                        }
                    }else if num4 == 3{
                    println!("O saldo da conta{} e de: {}",acesso,conta[acesso as usize]);
                    }else if num4 == 4{
                        println!("Voce saiu com sucesso");
                        saida2 = 1;
                    }else{
                        println! ("Valor invalido");
                    }
                }
            }else{
                println!("Nao existe essa conta!");
            }
    }
    else if num2 == 3{
        println!("Voce saiu com sucesso!");
        saida0 = 1;
    }
    else{
        println! ("Valor invalido");
    }
}
}
```


Figura 4.6: Código Rust: Estruturas de repetição, vetores e ponteiros

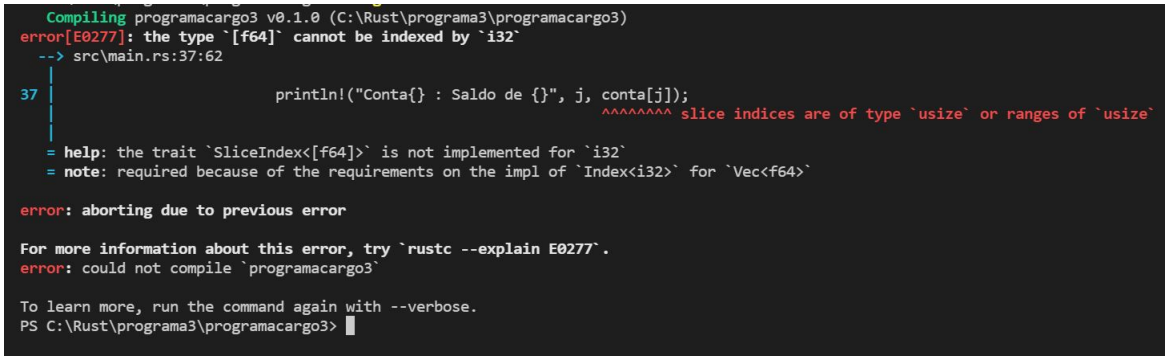

Figura 4.7: Exemplo 3 da interface dos erros

`https://github.com/DaniloFukuda/TCC-ICC-DaniloFukuda/tree/Codigo-em-Rust/programa3`.

Tal código cria um vetor, que cada posição do vetor será o numero da conta, e o valor no vetor será o saldo da conta. Para otimizar o uso do código o número da conta passou a ser o espaço ocupado no vetor, o saldo da conta se tornou o valor da variável do vetor.

No código em Rust, como visto na Figura 4.6, as primeiras diferenças observadas em Rust e em C no uso de estruturas de repetição e vetor, foi a criação de um Vetor, com o seguinte comando ":Vec<f64> = Vec::new()", que cria um vetor do tipo float com 64 bits, e a criação de um espaço no Vetor é feita pelo comando push, mostrada na linha 51 do código exposto na figura

Outra diferença é o comando "as"visto na linha 100 do código, em que a variável acesso precisa ser associada uma variável do tipo usize.

E o usize é o tipo inteiro não assinado de tamanho de ponteiro, o tamanho dessa primitiva é a quantidade de bytes que são necessários para fazer referência a qualquer local na memória. Por exemplo, em um destino de 32 bits, isso é 4 bytes e em um destino de 64 bits, isso é 8 bytes, tal explicação está exposta no site `https://doc.rust-lang.org/std/primitive.usize.html`.

Observando o tratamento de erros no código 3, como visto na Figura 4.6, a linguagem Rust novamente tem pontos positivos para o uso em turmas introdutórias de programação. Por exemplo, na linha 37, ao tirar a palavra as, o código ficaria assim: "println!("Conta : Saldo de ", j, conta[j]);", em que essa linha de comando estaria mostrando o numero da conta e seu saldo, dentro de um loop que percorreria o vetor, porém aparece o erro com o seguinte nome:"the type ‘[f64]‘ cannot be indexed by ‘i32‘", e com o seguinte comentário: "slice indices are of type ‘usize‘ or ranges of ‘usize‘". Como foi explicado acima, para acessar um determinado vetor, é necessário que seu índice seja do tipo de variável usize, portanto, para corrigir esse erro é necessário mudar para println!("Conta : Saldo de ", j, conta[j as usize]);", assim o código consegue acessar o índice indicado. Tudo isso é

apresentado na Figura 4.7.

Ter consciência de detalhes como este e sua implicação na correção e na segurança do software resultante disso é um aspecto que deve ser levado em consideração na formação dos futuros profissionais da área de computação.

4.5 Metodologia para comparar a linguagem de programação C e Rust

Para formalizar a comparação entre a linguagem de programação C e Rust foi utilizada a metodologia apresentada por Linda Mannila em seu artigo com o seguinte título: "An Objective Comparison of Languages for Teaching Introductory Programming". [18]

Tal artigo tem como objetivo fazer uma comparação objetiva de linguagens comuns, com base nas decisões de design usadas por proeminentes criadores de ensino de línguas, tirando conclusões que permitem instrutores tomarem decisões para aplicarem nos cursos introdutórios.

Os dezessete critérios de comparação foram criados a partir dos próprios critérios utilizados pelos criadores de línguas que são consideradas "línguas de ensino"(teaching languagens). Tais criadores são:

- Seymour Papert (criador do LOGO)
- Niklaus Wirth (criador do Pascal)
- Guido van Rossum (criador do Python)
- Bertrand Meyer (criador do Eiffel)

Os critérios são divididos em três principais, sendo eles divididos em subcritérios. Não será detalhado cada um dos critérios, tendo em vista que o artigo de Linda Mannila faz um ótimo trabalho explicando cada um deles, e também porque os critérios e subcritérios são bem claros. A avaliação da linguagem C foi transcrita do artigo da Linda Mannila e a avaliação do Rust foi feita através dos conhecimentos adquiridos na revisão sistemática da literatura e a comparação anterior feita através dos códigos de ambas linguagens de programação.

Assim seguindo os critérios para fazer uma comparação avaliativa e objetiva entre Rust e C, foi feita a tabela Tabela 4.1.

E percebe-se que a linguagem C teve 8 critérios atendidos, enquanto a linguagem Rust teve 14, podendo assim ser feita uma comparação mais objetiva entre as duas linguagens.

O subcritério "não é um exemplo do fenômeno QWERTY"pode gerar mais dúvidas do que outros subcritérios analisados, então vale a pena explicar o porque de marcar este

Tabela 4.1: Comparação por critérios entre Rust e C.

Critérios	**C**	**Rust**
Aprendizagem		
É adequado para o ensino		X
Pode ser usado para aplicar analogias físicas		X
Oferece um framework genérico	X	X
Promove um design para o ensino de software		X
Design e Ambiente		
É interativo e facilita o desenvolvimento rápido de código		X
Promove a escrita de programas corretos		X
Permite que os problemas sejam resolvidos em pedaços	X	X
A linguagem fornece um ambiente de desenvolvimento intuitivo		X
Suporte e disponibilidade		
Tem uma comunidade de usuários que oferece suporte	X	X
É open source para que qualquer um possa contribuir com o seu desenvolvimento		X
Tem suporte consistente em todos os ambientes	X	X
Está livre e facilmente disponível	X	X
É apoiado por um bom material de ensino		X
Além da programação introdutória		
Não é usado apenas na educação	X	X
É extensível	X	X
É confiável e eficiente	X	X
Não é um exemplo do fenômeno QWERTY		X
Pontuação dos Critérios dos Autores	**8**	**17**

subcritério na tabela, pois o fenômeno QWERTY é quando se utiliza uma linguagem de programação há muito tempo e assim a linguagem é utilizada somente por isso, sem questionar quais pontos fortes tal linguagem possa ter, assim o Rust não é um exemplo deste fenômeno, pois é uma linguagem nova e assim não é utilizada só porque todo mundo usa e está sendo utilizada no mercado.

Capítulo 5

Conclusão

Tendo em vista que o objetivo do primeiro curso de programação de computadores é fornecer conceitos e conhecimentos para os alunos compreenderem o fundamental dos constructos da programação de tal forma que devem ser capazes de programar a solução um determinado problema [1], faz sentido utilizar uma linguagem que permita ao aluno se concentrar no que é importante para aprender a programar do que nos detalhes sintáticos da linguagem de programação.

Portanto a escolha da Primeira Linguagem de Programação(First Programming Language - FPL) que irá ser usada em um curso de programação introdutório é muito importante para o desenvolvimento dos alunos, visto que cada linguagem de programação tem suas vantagens e desvantagens no seu uso.

No artigo "A cursory overview of the Rust programming language" [15] encontrado na revisão sistemática feita neste trabalho, foi destacado uma entrevista feita pelo site InfoQ, um site bem influente que aborda vários assuntos da programação, em que o engenheiro de software Graydon Hoare, que começou a trabalhar na criação da linguagem Rust em 2006, aborda na entrevista que o Rust tem recursos bem interessantes para os programadores, sendo eles:

- Correspondência de padrões e tipos de dados algébricos (enums)
- Simultaneidade baseada em tarefas.
- Tarefas leves podem ser executadas em paralelo sem compartilhar nenhuma memória.
- Funções de ordem superior. (fechamentos)
- Polimorfismo, combinando interfaces, semelhantes a Java e classes de tipo Haskell Genéricos.
- Sem estouro de buffer.

- Imutável por padrão.
- Um coletor de lixo sem bloqueio.
- Segurança de memória garantida.
- Threads sem corridas de dados.
- Tempo de execução mínimo.

Comparando Rust com a linguagem C apresentados nas imagens 4.1, 4.3, 4.5 e 4.6, tais diferença na sintaxe de C e Rust, percebe-se que a linguagem Rust é bem mais rigorosa com o tratamento das variáveis. Isso pode dificultar o entendimento dos alunos iniciantes de programação, porém tais obstáculos já seriam necessários ser vencidos para que eles consigam o objetivo final, que é fornecer conceitos e conhecimentos para os alunos compreenderem o fundamental dos constructos da programação. Então melhor que se utilizem de uma ferramenta melhor, que oferece ferramentas de desenvolvedor para facilitar o entendimento e a correção do código, como por exemplo as ferramentas Cargo, Rustfmt e Rust Language Server, apresentadas no capítulo anterior.

Um exemplo de utilização dessas ferramentas do Rust seria os comentários dos erros apresentados no código, que aparecem logo após a inicialização do programa, pois assim o aluno poderia identificar seus erros e corrigi-los de forma bem mais rápida em comparação com a linguagem C, já que o mesmo apresenta comentários bem vagos sobre os erros do código.

Tais recursos ajudariam programadores iniciantes a programar de forma mais coerente e segura, pois a linguagem Rust é fortemente tipada, sendo assim uma boa escolha para ser a Primeira Linguagem de Programação para um curso introdutório de programação.

Outro argumento favorável ao uso de Rust nas matérias introdutórias seria a metodologia usada no curso Introdução à Computação Paralela e Distribuída da USP, que produziu uma monografia com o título: "Um panorama sobre Rust" [13], selecionado a partir da revisão sistemática da literatura feita neste trabalho, que seria um exemplo pratico de seu uso em uma matéria de programação.

Portanto com este trabalho que apresenta uma revisão sistemática da literatura sobre o uso de Rust nas matérias introdutórias de programação, e uma comparação da linguagem C e Rust em códigos apresentados neste mesmo, são expostos vários argumentos, sendo eles em literatura ou nos exemplos dos códigos apresentados, que favorecem a utilização da linguagem Rust no ensino de APC, na qual tem se utilizado das linguagens C e Python para seu ensino e se fosse aprovado tal mudança, seria necessário atualizar diversas disciplinas dos cursos do CIC, como por exemplo,Ciência da Computação, Computação(licenciatura), Engenharia Mecatrônica, Engenharia de Computação, Engenharia Mecatrônica e Engenharia Elétrica, o que não é uma decisão simples.

Ao analisar todos os pontos apresentados, é observado que a linguagem Rust tem ganhado notoriedade de forma geral, seja no campo acadêmico, como no desenvolvimento de novos programas, como por exemplo, na plataforma de criação de aplicativos do Android, dado seus pontos fortes que foram apresentados neste trabalho, portanto, existe muitas possibilidades tanto na parte acadêmica, como na parte de desenvolvimento de sistemas mais seguros, que precisam ser exploradas para novas descobertas e conclusões serem feitas.

Outro ponto importante para acrescentar nesta conclusão seria a comparação objetiva feita no capítulo anterior, seguindo os critérios de Linda Mannila em seu artigo "An Objective Comparison of Languages for Teaching Introductory Programming" [18]. Em que neste artigo é feita uma comparação avaliativa e objetiva entre as linguagens de programação por vários critérios que são importantes para o ensino de programação. Utilizando os critérios apresentados neste artigo, a linguagem Rust teve uma pontuação de 17, ou seja, 17 critérios foram atendidos, enquanto a linguagem C teve somente 8 critérios atendidos. Para ilustrar melhor tais critérios, eles estão apresentados na Tabela 4.1.

Em um cenário em que as ameaças cibernéticas são crescentes, em que as notícias de roubo de dados, erros no código de sistemas de operacionais e etc. causam prejuízos crescentes à sociedade, ameaçando inclusive nossa segurança pessoal, talvez a modernização dos cursos da área de computação por meio do ensino de programação usando a linguagem Rust seja uma novidade necessária. Mas para isto, é necessário experimentar e avaliar o Rust como uma primeira linguagem de programação em um curso introdutório de programação, o que já seria um trabalho futuro.

Referências

[1] Farooq, Muhammad Shoaib, Sher Afzal Khan, Farooq Ahmad, Saeed Islam e Adnan Abid: *An evaluation framework and comparative analysis of the widely used first programming languages*. PloS one, 9(2):e88941, 2014. 1, 33

[2] Sobral, Sonia: *30 years of cs1: Programming languages evolution*. novembro 2019. 2

[3] Bogdanchikov, A, M Zhaparov e R Suliyev: *Python to learn programming*. Journal of Physics: Conference Series, 423:012027, apr 2013. `https://doi.org/10.1088/1742-6596/423/1/012027`. 2

[4] Rubiano, Sandra Milena Merchán: *Teaching computer programming*. Education and Information Technologies, 7, March 2002. 2

[5] Wing, Jeannette M.: *Computational thinking and thinking about computing*. Phil. Trans. R. Soc. A, 366, july 2008. 5

[6] Jenkins, Tony. *In Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*, 4, August 2002. 5

[7] Sobral, Sonia: *The First Programming Language and Freshman Year in Computer Science: Characterization and Tips for Better Decision Making*, páginas 162–174. maio 2020, ISBN 978-3-030-45696-2. 6

[8] Allberg, Filip, Adam Dahlgren Lindström, Patrik Hörnquist, Jakob Lindqvist e Emil Marklund: *A cursory overview of the Rust programming language*. Programming Languages, página 10. 7, 12, 14

[9] Okoli, Chitu e Kira Schabram: *A guide to conducting a systematic literature review of information systems research*. 2010. 9, 10, 11, 13

[10] Theoktisto, Victor: *A Functional Paradigm using the C Language for Teaching Programming for Engineers*. Em *2018 XLIV Latin American Computer Conference (CLEI)*, páginas 820–828, São Paulo, Brazil, outubro 2018. IEEE, ISBN 978-1-72810-437-9. `https://ieeexplore.ieee.org/document/8786354/`, acesso em 2020-10-26. 12, 15

[11] Röijezon, Teo Klestrup e Therese Kennerberg: *A Physical Platform For Teaching Distributed Systems In A Simplified Setting*. página 31. 12, 16

[12] Electronics and IT Unit, École Centrale des Arts et Métiers (ECAM), Sébastien Combéfis, Saïkou Ahmadou Barry, Electronics and IT Unit, École Centrale des Arts et Métiers (ECAM), Martin Crappe, Electronics and IT Unit, École Centrale des Arts et Métiers (ECAM), Mathieu David, Electronics and IT Unit, École Centrale des Arts et Métiers (ECAM), Guillaume de MOFFARTS, Electronics and IT Unit, École Centrale des Arts et Métiers (ECAM), Hadrien Hachez, Electronics and IT Unit, École Centrale des Arts et Métiers (ECAM), Julien Kessels e Electronics and IT Unit, École Centrale des Arts et Métiers (ECAM): *Learning and Teaching Algorithm Design and Optimisation Using Contests Tasks*. OI, 11(1):19–28, julho 2017, ISSN 18227732. `http://www.ioinformatics.org/oi/shtm/v11_2017_19_28.shtml`, acesso em 2020-10-26. 12

[13] Koch, Thilo e de Junho de: *Monograa na matéria MAC 5742: Introdução à Computação Paralela e Distribuída Professor Alfredo Goldman Vel Lejbman IME USP*. página 12. 12, 18, 34

[14] Crichton, Will: *From Theory to Systems: A Grounded Approach to Programming Language Education*. arXiv:1904.06750 [cs], abril 2019. `http://arxiv.org/abs/1904.06750`, acesso em 2020-10-26, arXiv: 1904.06750. 12, 17

[15] Johnson, Daniel, Sebastian Deterding, Kerri Ann Kuhn, Aleksandra Staneva, Stoyan Stoyanov e Leanne Hides: *Gamification for health and wellbeing: A systematic review of the literature*. Internet Interventions, 6:89 – 106, 2016, ISSN 2214-7829. `http://www.sciencedirect.com/science/article/pii/S2214782916300380`. 13, 33

[16] JIANG Hui, LIN Dong, ZHANG Xingyuan: *"type system in programming languages "*. "Journal of Computer Science and Technology", "16"("3"):"286–292", "2001". 15

[17] Walker, David: *Substructural type systems*. Advanced topics in types and programming languages, páginas 3–44, 2005. 15

[18] Mannila, Linda e Michael de Raadt: *An objective comparison of languages for teaching introductory programming*. Em *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*, Baltic Sea '06, página 32–37, New York, NY, USA, 2006. Association for Computing Machinery, ISBN 9781450378383. `https://doi.org/10.1145/1315803.1315811`. 30, 35