Programação Imperativa

Pedro Horchulhack

Quem sou eu?
Pedro Horchulhack

Formação:

- Bacharel em Ciência da Computação - PUCPR
- Mestre em Informática - PUCPR
- Doutorado em informática - PUCPR (andamento)

Áreas de atuação:

- Sistemas distribuídos
- Aprendizagem de máquina
- Detecção de Intrusão baseada em Rede
- Virtualização

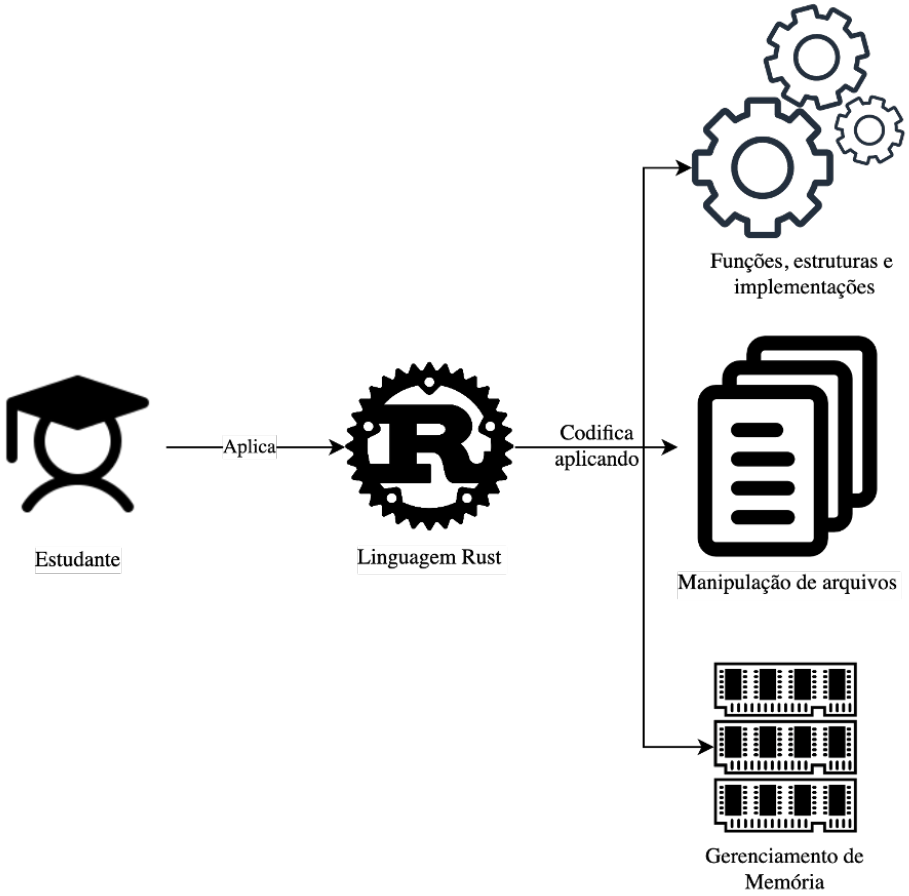
Plano de Ensino

Temas de Estudo

1. Tipos de dados: primitivos e compostos
2. Estruturas de controle de fluxo
3. Laços de repetição
4. Gerenciamento de memória
5. Estruturas e implementações
6. Programação segura

Plano de Ensino

Mapa Mental



Plano de Ensino

Resultados de Aprendizagem

- **RA1:** Codificar programas utilizando as construções fundamentais de programação e algoritmos na linguagem Rust (50%).

- **RA2:** Codificar com utilização de gerenciamento de memória com alocação e liberação dinâmica de memória (50%).

Plano de Ensino

Metodologia e avaliação

- **Formativo**:
  - Exercícios de programação
  - Feedbacks individuais e em grupo

- **Somativo**:
  - Avaliações individuais sobre o papel (60%)
  - Trabalho Discente Efetivo (TDE) (40%)

Plano de Ensino

Cronograma

Semana	RAs	Atividades pedagógicas
1	1	Revisão de fundamentos de lógica de programação
2		Tipos de variáveis primitivas e estruturas de controle de fluxo
3		Laços de repetição
4		Funções, passagem de parâmetros por valor e por referência
5		Tipos enumerados, vetores e arrays
6		Estruturas e implementações
7		Recursividade
8		Somativa RA1
9	-	Semana acadêmica
10	2	Gerenciamento de memória, referências e ponteiros
11		
12		Ownership e Borrowing
13		
14		Smart pointers
15		Módulos e *crates*
16	1	Somativa RA2 + Recuperação RA1
17	2	Recuperação RA2 + Feedback

Plano de Ensino

Cronograma

Quadro 7-1. Datas previstas para as avaliações somativas e recuperações, podendo sofrer alterações de acordo com necessidades.

Somativas	Data Prevista
Prova RA1	19/09/2024
Prova RA2	14/11/2024
Recuperação RA1	14/11/2024
Recuperação RA2	21/11/2024

Plano de Ensino

Bibliografia

Básica:

· FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação: a construção de algoritmos e estruturas de dados com aplicações em Python**. 4. ed. São Paulo: Grupo A, 2022. E-book. Disponível em: https://plataforma.bvirtual.com.br. Acesso em: 20 jul. 2023.

· ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. **Fundamentos da programação de computadores:** algoritmos, Pascal, C/C++ e Java. 2. ed. São Paulo: Pearson Prentice Hall, 2007. viii, 434 p. ISBN 978-85-7605-148-0 (broch.).

· SOFFNER, Renato K. **Algoritmos e Programação em Linguagem C, 1ª edição**. São Paulo: Editora Saraiva, 2013. *E-book.* ISBN 9788502207530. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788502207530/

Complementar:

· DEITEL, H.; DEITEL, P. **Como Programar Em C**. Tradução: Amir Kurban. 2. ed. Rio de Janeiro: LTC, 1999. p. 1–486.

· JUNG, Ralf et al. RustBelt: **Securing the foundations of the Rust programming language**. Proceedings of the ACM on Programming Languages, v. 2, n. POPL, p. 1-34, 2017.

· STROUSTRUP, Bjarne. **A linguagem de programação C++**. Bookman, 2000.

· DAMAS, Luís. **Linguagem C.** 10. Rio de Janeiro: LTC, 2006. 1 recurso online. ISBN 9788521632474.

· MORAIS, Izabelly S.; LEON, Jeferson F.; SARAIVA, Maurício O.; et al. Algoritmo e programação - Engenharia. Grupo A, 2018. E-book. ISBN 9788595024731. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595024731/. Acesso em: 06 nov. 2023.

· CORMEN, Thomas. **Algoritmos - Teoria e Prática**. Grupo GEN, 2012. *E-book.* ISBN 9788595158092. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595158092/. Acesso em: 06 nov. 2023

Dúvidas?

Programação Imperativa
Breve contexto

- Também conhecida como programação procedural
  - Baseia-se na ideia de procedimentos executados em sequência

- Nós pedimos ao computador o que ele deve fazer
  - Some A com B, subtraia B de A *etc*.

- Descrevemos *como* o programa deve operar

Programação Imperativa

Breve contexto

- Estamos preocupados em como o programa irá executar as instruções ou declarações e de que forma isso afetará a memória!

Programação Imperativa

Breve contexto — Como podemos codificar?

1. Entender o problema
2. Esboçar uma solução geral
3. Decompor a solução em componentes
4. Desenvolver uma solução específica para cada componente
5. Testar se a solução adotada por cada componente é valida e correta
6. **Implementar a solução em uma linguagem de programação**

Programação Imperativa

Breve contexto — Como podemos codificar?

1. Entender o problema
2. Esboçar uma solução geral
3. Decompor a solução em componentes
4. Desenvolver uma solução específica para cada componente
5. Implementar a solução em uma linguagem de programação
6. Testar se a solução adotada por cada componente é valida e correta

Programação Imperativa
Breve contexto

- Muitas linguagens são procedurais/imperativas
  - C
  - Rust
  - FORTRAN
  - Assembly
  - Python
  - …

Programação Imperativa
Breve contexto

- Muitas linguagens são procedurais/imperativas
  - C
  - **Rust** ←
  - FORTRAN
  - Assembly
  - Python
  - …

Linguagem Rust

- Surgiu em 2010
- Desenvolvida pela Mozilla Research
- Projetada com o objetivo de ser mais segura do que as demais linguagens
- Estabilidade e velocidade

Linguagem Rust

Amazon explica por que usa a linguagem de programação Rust em seus projetos

Por Dácio Castelo Branco | Editado por Claudio Yuge | 23

PRESS RELEASE | Dec. 6, 2023

U.S. and International Partners Issue Recommendations to Secure Software Products Through Memory Safety

Rust developers at Google are twice as productive as C++ teams

Code shines up nicely in production, says Chocolate Factory's Bergstrom

Thomas Claburn Sun 31 Mar 2024 // 16:33 UTC

Linguagem Rust

- Áreas de atuação com Rust
  - Sistemas operacionais
  - Blockchain
  - Gestão de memória
  - Jogos
  - Sistemas embarcados

Revisão

- Lista de exercícios no AVA

Ambiente de desenvolvimento

Ambiente de desenvolvimento

- Estaremos utilizando a IDE RustRover, da JetBrains

- Acesse e baixe por este link: https://www.jetbrains.com/rust/download/

Linguagem Rust

Variáveis

- Por padrão, as variáveis na linguagem Rust são **imutáveis**, declaradas da seguinte maneira:

```
let x = 5;
```

- Qual o tipo da variável declarada acima?

Linguagem Rust

Variáveis

- Devido as variáveis serem imutáveis, se tentarmos executar o seguinte código Rust, ele não irá compilar:

```
fn main() {
    let x = 5;

    println!("O valor de x é {x}");

    x = 5 * 2;
    println!("Agora o valor de x é {x}");
}
```

Linguagem Rust

Variáveis

- E quando quisermos alterar o valor de uma variável, o que fazemos?

```
let mut x = 5;
```

Linguagem Rust

Tipagem

- É uma linguagem fortemente tipada, ou seja, não podemos alterar os tipos das variáveis após definirmos o seu tipo ou após a inferência

- Exemplos de inferência e declaração:

```
fn main() {
    let mut x = 5; // Inferência de tipo
    let y: i32 = 11; // Declaração do tipo
}
```

Linguagem Rust

Tipagem

- O que acontece se tentarmos alterar o tipo de uma variável?

```
fn main() {
    let mut x = 5; // Inferência de tipo
    let y: i32 = 11; // Declaração do tipo

    x = "Olá!";
}
```

Linguagem Rust

Tipagem

- O que acontece se tentarmos alterar o tipo de uma variável?

```
fn main() {
    let mut x = 5; // Inferência de tipo
    let y: i32 = 11; // Declaração do tipo

    x = "Olá!";
}
```

```
error[E0308]: mismatched types
  --> src/main.rs:30:9
   |
27 |     let mut x = 5; // Inferência de tipo
   |                 - expected due to this value
...
30 |     x = "Olá!";
   |         ^^^^^^ expected integer, found `&str`
```

Tipos de variáveis

Tipos de variáveis

Tipos primitivos

- Lidamos com dados a todo momento como, por exemplo:
  - Nomes
  - Datas
  - Dias de aniversário
  - Velocidade
  - Comprimento
  - ...

Tipos de variáveis

Tipos primitivos

- Em programação não é diferente, pois manipulamos dados de maneira que o programa se comporte da maneira desejada

- Um exemplo pode ser a calculadora do nosso *smartphone*
  - Movemos e processamos dados para que os cálculos sejam realizados e para que obtenhamos o resultado esperado

Tipos de variáveis

Tipos primitivos

- Para a construção de algoritmos, temos quatro tipos básicos de dados primitivos, ou seja, o tipo de dado mais básico (FORBELLONE, EBERSPÄCHER, 2022):
  - Inteiro
  - Real
  - Caracter
  - Lógico

Tipos de variáveis

Tipos primitivos

- Números inteiros: (-3, -2, 0, 1, 34, 100, …)
  - Qualquer número que esteja no conjunto dos inteiros, incluindo o 0 (zero)
  - Exemplos:
    - Eu tenho **22** anos de idade
    - Meu notebook possui **16**GB de memória RAM
    - Minha garrafa d’água suporta até **2** litros

Tipos de variáveis

Tipos primitivos

- Números reais: (-15.3, -0.03, 0.5, 1.32, 17.3, …)
  - “Números com vírgula”
  - Exemplos:
    - Um pão de queijo no centro de Curitiba custa R$ **3,50**
    - O monitor tem **14,5** polegadas
    - Minha altura é **1,70** metros

Tipos de variáveis

Tipos primitivos

- Caracter:
  - Corresponde à caracteres individuais, sendo numéricos (0 … 9), alfabéticos (A…Z, a…z) e especiais (!, @, #, ?, *, …)

- Strings são conjuntos de caracteres

Tipos de variáveis

Tipos primitivos

- Lógico:
  - Qualquer informação que pode assumir apenas duas situações (FORBELLONE, EBERSPÄCHER, 2022)

  - Exemplo:
    - A porta pode estar **aberta** ou **fechada**
    - A lâmpada pode estar **ligada** ou **desligada**

Tipos de variáveis

Tipos primitivos

- Devido as variáveis serem imutáveis, se tentarmos executar o seguinte código Rust, ele não irá compilar:

```
fn main() {
    let x = 5;

    println!("O valor de x é {x}");

    x = 5 * 2;
    println!("Agora o valor de x é {x}");
}
```

Tipos de variáveis

Tipos primitivos

- Devido as variáveis serem imutáveis, se tentarmos executar o seguinte código Rust, ele não irá compilar:

```rust
fn main() {
    let x = 5;

    println!("O valor de x é {x}");

    x = 5 * 2;
    println!("Agora o valor de x é {x}");
}
```

```
  --> src/main.rs:26:9
   |
26 |     let x = 5;
   |         ^ help: if this is intentional, prefix it with an underscore: `_x`
   |
   = note: `#[warn(unused_variables)]` on by default

error[E0384]: cannot assign twice to immutable variable `x`
  --> src/main.rs:35:5
   |
31 |     let x = 5;
   |         -
   |         |
   |         first assignment to `x`
   |         help: consider making this binding mutable: `mut x`
...
35 |     x = 5 * 2;
   |     ^^^^^^^^^ cannot assign twice to immutable variable

For more information about this error, try `rustc --explain E0384`.
```

Tipos primitivos em Rust

Inteiros

Tamanho	Assinado	Não-assinado
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Tipos primitivos em Rust

Inteiros

- O que significa esse tamanho?

- Tais tamanhos assumem tamanhos pré-definidos por conta da memória, ou seja, quanto espaço a variável ocupará

Tamanho	Assinado	Não-assinado
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Tipos primitivos em Rust

Inteiros

- Como podemos saber o tamanho mínimo e máximo de cada tipo de variável assinada?

- Tamanho mínimo: $-2^{(n-1)}$
- Tamanho máximo: $2^{(n-1)} - 1$
  - n: tamanho da variável

Tamanho	Assinado	Não-assinado
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Tipos primitivos em Rust

Inteiros

- Como podemos saber o tamanho mínimo e máximo de cada tipo de variável assinada?

- Tamanho mínimo: -128
- Tamanho máximo: 127
  - $n = 8$

Tamanho	Assinado	Não-assinado
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Tipos primitivos em Rust

Inteiros

- Como podemos saber o tamanho mínimo e máximo de cada tipo de variável **não-assinada**?

- Tamanho mínimo: 0
- Tamanho máximo: $2^n - 1$
  - n: tamanho da variável

Tamanho	Assinado	Não-assinado
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Tipos primitivos em Rust

Inteiros

- Como podemos saber o tamanho mínimo e máximo de cada tipo de variável **não-assinada**?

- Tamanho mínimo: 0
- Tamanho máximo: 255
  - $n = 8$

Tamanho	Assinado	Não-assinado
8-bit	`i8`	`u8`
16-bit	`i16`	`u16`
32-bit	`i32`	`u32`
64-bit	`i64`	`u64`
128-bit	`i128`	`u128`
arch	`isize`	`usize`

Tipos primitivos em Rust

Reais

- Em Rust, temos somente dois tamanhos de tipo de variável de ponto flutuante

- Seguem o padrão IEEE-754

- Por padrão, em Rust variáveis de ponto flutuante são declaradas como `f64` se não especificarmos diretamente o **tipo**

Tamanho	Assinado
32-bit	`f32`
64-bit	`f64`

Tipos primitivos em Rust

Caracter

- O tipo caracter em Rust possui o tamanho de 4 bytes de tamanho (32-bits)

```
fn main() {
    let c = 'z';
    let z: char = 'ℤ';
    let heart_eyed_cat = '😻';
}
```

Tipos primitivos em Rust

Booleano

- O tipo booleano pode assumir dois valores: verdadeiro e falso (true, false)

```
fn main() {
    let porta_aberta = false;
    let lampada_acesa = true;
}
```

Tipos primitivos em Rust

Exemplo de código para adivinhar a idade de alguém

```rust
fn main() {
    println!("Por favor, digite sua idade: ");
    let mut idade = String::new();

    // Lendo entrada do usuário
    io::stdin().read_line(&mut idade).expect("Erro ao ler linha");

    // Convertendo o valor da variável 'idade' em número inteiro de 32 bits
    let idade_num: i32 = idade.trim()
                              .parse()
                              .expect("Erro ao converter número");

    println!("Sua idade é {idade_num} anos”);
}
```

Exercícios

1. Escreva um algoritmo em Rust para calcular a idade de alguém, sabendo seu ano de nascimento

2. Escreva um algoritmo em Rust para calcular o valor, em reais, que deve ser pago para um cliente de uma locadora de carros. Sabe-se que:
    1. O valor da locação para cada carro é 100.00 reais;
    2. O cliente pode locar um único carro por vários dias;

3. Leia a temperatura do teclado em Celsius e imprima o equivalente em Fahrenheit

$$C = (F - 32) * \frac{5}{9}$$