Paradigmas de Programação

**Introdução à disciplina**

## Introdução

Este material contém as notas de aula iniciais da disciplina *Paradigmas de Programação*. O objetivo é apresentar a disciplina, resgatar o histórico e as principais características de Linguagens de Programação.

## Linguagem de Programação

Um dos conceitos mais importantes para se aprender sobre Paradigmas de Programação é o de Linguagem. Segundo o dicionário Oxford (2024), a linguagem é “*qualquer meio sistemático de comunicar ideias ou sentimentos através de signos convencionais, sonoros, gráficos, gestuais etc*”. No universo da programação, linguagens de programação são utilizadas para instruir os computadores sobre como executar tarefas específicas. Ao desenvolver um código para uma linguagem de programação, o programador comunica suas ideias por meio de instruções, que devem ser escritas utilizando símbolos conhecidos e regras gramaticais que permitem que o computador compreenda o código, ou seja, permitem a conversão do algoritmo (a sequência de passos escrita pelo programador) em um programa.

Existem diversas linguagens de programação. Cada uma com suas próprias características, paradigma(s) e aplicações específicas. Apesar disso, toda linguagem de programação possui alguns elementos básicos em comum. Entre eles:

* Instruções (exemplo: print, na linguagem Python)
* Palavras e símbolos reservados (exemplo: =, em linguagens como Python e C)
* Regras (exemplo: while (expressão lógica), em linguagens como C e C++)
* Significado
* Sistema de tipos

## Razões para Estudar Conceitos de Linguagens de Programação

Sebesta (2012) afirma que é comum as pessoas acharem que aprender uma ou duas linguagens de programação é suficiente. Uma possível explicação para isso é o fato de que o cientista da computação tem diversos temas relevantes a estudar. Apesar disso, o autor cita benefícios de estudar linguagens de programação:

* incrementar a capacidade de expressar ideias, natural ao aprofundar os conhecimentos;
* incrementar o conhecimento para escolher linguagens apropriadas;
* incrementar habilidades pessoais para aprender novas linguagens. Por exemplo: quem compreende os conceitos de programação funcional tende a ter mais facilidade para aprender Haskell e Miranda;
* incrementar o vocabulário técnico, facilitando o entendimento dos conceitos;
* conscientização sobre a importância da implementação, permite propor e usar funções otimizadas, identificar *bugs* etc;
* aprimorar o uso das linguagens já conhecidas, pois conceitos estudados podem ser aplicáveis nelas.

Em geral, a escolha pela(s) linguagem(ns) de programação a ser(em) estudada(s) envolve diversos fatores. Entre eles, pode-se citar:

* **Objetivos do Projeto**: a linguagem deve ser adequada para o propósito do projeto. Por exemplo, se o objetivo é desenvolver um aplicativo web, linguagens como JavaScript, Python ou Ruby podem ser mais adequadas.
* **Facilidade de Aprendizado**: essa característica é particularmente útil para iniciantes. Algumas linguagens são conhecidas por serem mais fáceis de aprender do que outras, sendo um caminho mais curto para quem está iniciando os estudos em linguagens de programação.
* **Comunidade e Suporte**: a disponibilidade de recursos de aprendizado (apostilas, vídeos, livros etc.), documentação e suporte da comunidade pode influenciar a escolha da linguagem. Uma comunidade ativa pode facilitar o aprendizado e fornecer suporte durante o desenvolvimento do projeto.
* **Ferramentas Disponíveis**: o ecossistema de uma linguagem, incluindo bibliotecas, *frameworks* e ferramentas de desenvolvimento disponíveis, pode influenciar a escolha da linguagem por facilitar o desenvolvimento e melhorar a produtividade.
* **Desempenho e Eficiência**: algumas linguagens são mais eficientes em termos de uso de recursos computacionais do que outras. Se esse for um fator crítico para a aplicação que se deseja desenvolver, linguagens como C e C++ são frequentemente escolhidas devido à sua capacidade de gerenciar recursos de forma eficiente e oferecer controle de baixo nível sobre o hardware do sistema.
* **Demanda do Mundo de Trabalho**: a quem busca oportunidades de trabalho, as linguagens mais usadas profissionalmente podem influenciar a escolha da linguagem a ser estudada.

O [índice TIOBE](http://www.tiobe.com/tiobe-index/index.htm) (TIOBE, 2024), por exemplo, é um indicador de popularidade de linguagens de programação, atualizado mensalmente. A [StackOverflow](https://stackoverflow.com/), popular plataforma de perguntas e respostas, divulga anualmente relatórios com os resultados de suas pesquisas sobre tecnologias. Entre elas, resultados sobre as linguagens de programação, de *script* e de marcação mais amadas, temidas e desejadas. Esses resultados são divulgados anualmente e podem ser acompanhados nos links abaixo:

* <https://insights.stackoverflow.com/survey/2019#most-loved-dreaded-and-wanted>
* <https://insights.stackoverflow.com/survey/2020#most-loved-dreaded-and-wanted>
* <https://insights.stackoverflow.com/survey/2021#technology-most-loved-dreaded-and-wanted>
* <https://survey.stackoverflow.co/2022#technology-most-loved-dreaded-and-wanted>

A partir do relatório de 2023, a StackOverflow passou a trazer dados sobre [linguagens mais admiradas e desejadas](https://survey.stackoverflow.co/2023/#technology-admired-and-desired).

## Domínios de Programação

Existe uma grande variedade de áreas de aplicações de Computação. Algumas delas são:

* **Aplicações Científicas**: uso de métodos e técnicas científicas para investigar fenômenos naturais, explorar o funcionamento de sistemas complexos e desenvolver soluções para problemas de diversas áreas como física, química, biologia, astronomia, geologia e outras. Podem abranger coleta e análise de dados experimentais até a modelagem computacional e simulação de sistemas. Exemplos envolvem aplicações de sequenciamento genético, modelagem de ecossistemas, simulação de processos bioquímicos para o desenvolvimento de drogas e terapias, modelagem climática, processamento de imagens médicas etc.
* **Aplicações de Negócios/Empresariais**: refere-se a softwares desenvolvidos para atender às necessidades e operações de negócios em empresas, visando melhorar a eficiência, automatizar processos e gerenciar recursos. Exemplos conhecidos são os Sistemas de Gestão Empresarial (*Enterprise Resource Planning* - ERP), que integram diferentes processos de negócios, como contabilidade, recursos humanos e vendas, em um único sistema.
* **Inteligência Artificial – IA**: diz respeito a uma área da Computação que se concentra no desenvolvimento de sistemas que podem executar tarefas que normalmente exigiriam inteligência humana. Isso inclui aprendizado de máquina, processamento de linguagem natural, visão computacional, entre outros. Exemplos incluem os assistentes virtuais inteligentes, tais como a Siri da Apple e a Alexa da Amazon, que podem entender comandos de voz, responder perguntas e executar tarefas com base em instruções dos usuários.
* **Programação de Sistemas**: desenvolvimento do que é conhecido como *software de sistema* (SEBESTA, 2011), ou seja, software de baixo nível, que interage diretamente com o hardware do computador e fornece serviços fundamentais. Segundo Sebesta (2011), isso envolve o sistema operacional e todas as ferramentas de suporte à programação de um sistema de computação. Windows e Ubuntu são exemplos de sistemas operacionais.
* **Software para *Web***: domínio que engloba o desenvolvimento de softwares destinados a serem executados em navegadores *web* e servidores *web*. Isso inclui não apenas linguagens de programação, mas também de marcação, como XHTML (SEBESTA, 2011). Há diversos exemplos de aplicações desse tipo, incluindo plataformas de redes sociais, páginas de notícias, páginas de jogos, de artigos científicos etc.

Diversas linguagens de programação são capazes de resolver esses problemas. Sebesta (2011) cita algumas associadas aos domínios citados. Para aplicações científicas, cita as linguagens Fortran e Algol 60. Para aplicações empresariais, cita a linguagem COBOL. Para IA, LISP e Prolog. Para programação de sistemas, C, e para Software para Web, Java, JavaScript e PHP.

## Gerações de Linguagens

A depender do autor seguido, podem existir quatro, cinco ou seis gerações de linguagens. Neste material, seguir-se-á a abordagem que classifica as linguagens em seis gerações. Essas gerações costumam ser classificadas da seguinte forma:

### 1ª Geração – **Linguage**ns **de Máquina**

O programa é um arquivo binário. É totalmente dependente de arquitetura. Como consequência, as operações podem ser diferentes em arquiteturas distintas. Esse tipo de linguagem não é legível para humanos.

### 2ª Geração – Linguagens Intermediárias/de Montagem

As linguagens dessa geração são de baixo nível, ainda dependentes de arquitetura. A abreviação usada para esse tipo de linguagem é ASM, que significa *Assembly* ou *Assembly Language*. Esse tipo de linguagem é legível para humanos. Um exemplo de código ASM está no quadro abaixo:

| **section .text**  **global \_start**  **\_start:**  **mov edx, len**  **mov ecx, msg**  **mov ebx, 1**  **mov eax, 4**  **int 0x80**  **mov eax, 1**  **int 0x80**  **section .data**  **msg db 'Hello, world!',0xa**  **len equ $ - msg** |
| --- |

O código acima pode ser salvo em um arquivo de texto de nome [aula1.asm](https://drive.google.com/file/d/1Z-Oqbbg7Tm-4mZqGjAbgegVuo2cLKA8K/view?usp=drive_link). A execução do código é feita no terminal, digitando o comando abaixo a partir do diretório em que o arquivo aula1.asm foi salvo:

nasm -f elf64 aula1.asm; ld aula1.o -o aula1; ./aula1

### 3ª Geração – Linguagens de Propósito Geral

Uma Linguagem de Propósito Geral atua em vários domínios de aplicação. Exemplos de linguagens de terceira geração: C, Java, Pascal, Python etc. Um exemplo de código para linguagem C está no quadro abaixo:

| **#include <stdio.h>**  **int main() {**  **printf("Hello, world!\n");**  **return 0;**  **}** |
| --- |

O código acima pode ser salvo em um arquivo de texto de nome [aula1.c](https://drive.google.com/file/d/15LIuYOMxTMH3OKEojIVoIOdOmMtrbb87/view?usp=drive_link). A execução do código é feita no terminal, digitando o comando abaixo a partir do diretório em que o arquivo aula1.c foi salvo:

**gcc -o aula1 aula1.c; ./aula1**

### 4ª Geração – Linguagens de Propósito Específico

As Linguagens de Propósito Específico atuam em algum domínio específico. Exemplos incluem as linguagens MATLAB, projetada para computação matemática e simbólica, R, projetada para análise estatística, e SQL, projetada para consulta estruturada a bancos de dados relacionais. O exemplo no quadro a seguir mostra o uso da linguagem R. Para acessar a ferramenta R, é necessário entrar no terminal e digitar R (a letra precisa ser maiúscula).

| **x <- c(1, 500, 98, 800)**  **mean(x)**  **sd(x)** |
| --- |

Na primeira linha, um vetor chamado x foi criado. Ele contém os valores 1, 500, 98 e 800. O operador <- é usado para atribuir valores a uma variável em R. O comando c() é usado para criar um vetor com os valores fornecidos. A segunda linha faz uma chamada à função mean com x como parâmetro. A função mean calcula a média dos valores do vetor. Após executá-la, a resposta obtida deve ser:

**[1] 349.75**

O valor [1] indica apenas que o resultado que o segue é o primeiro conjunto de valores impressos na saída. O valor 349.75 representa a média dos valores do vetor.

| **E o comando sd(x)? O que ele faz? Execute-o e descubra!** |
| --- |

Caso prefira, os comandos R podem ser escritos em um arquivo (exemplo: [aula1.R](https://drive.google.com/file/d/1kikTHrYcl6UN6yIbOE7cBaQxgCRKLK34/view?usp=sharing)) e executados no terminal através do comando abaixo:

**Rscript aula1.R**

### 

### 5ª Geração – Linguagens de Inteligência Artificial

São consideradas linguagens de quinta geração as linguagens lógicas e funcionais. Exemplos dessas linguagens são: Prolog, Lisp, Haskell, Cloujure e Miranda. Um exemplo de código para linguagem Haskell está no quadro abaixo:

| **main = putStrLn "Hello, World!"** |
| --- |

O código pode ser salvo em um arquivo de texto de nome [aula1.hs](https://drive.google.com/file/d/1_jZCW_q8dH_p8wa_jyo58PMtBqhkW3Hd/view?usp=sharing). A execução do código é feita no terminal, digitando o comando abaixo a partir do diretório em que o arquivo aula1.hs foi salvo:

**ghc aula1.hs; ./aula1**

Outra opção é abrir o terminal e digitar ghci. Esse comando executa o compilador interativo de Haskell. A ferramenta ghci estará em estado de prontidão quando estiver exibido a seguinte mensagem:

**GHCi, version 8.0.2: http://www.haskell.org/ghc/ :? for help**

**Prelude>**

Após exibir a mensagem, basta digitar diretamente o(s) comando(s) desejado(s). No exemplo de aula, o comando seria:

**putStrLn "Hello, World!"**

A linguagem Haskell e as ferramentas ghc e ghci serão abordadas em mais detalhes em aulas futuras, assim como a linguagem Prolog.

### 6ª Geração – Redes Neurais

Essa geração é caracterizada principalmente pelo foco em tecnologias de inteligência artificial e aprendizado de máquina, particularmente em relação às redes neurais. Essas linguagens são projetadas para simplificar o desenvolvimento de sistemas baseados em inteligência artificial, oferecendo abstrações de alto nível que permitem aos programadores expressar conceitos complexos de forma mais intuitiva. Além disso, são orientadas a dados e capazes de lidar com conjuntos de dados grandes e complexos, facilitando a análise e a modelagem de dados em larga escala. Exemplos incluem TensorFlow e Keras.

## Discussão

Propõe-se uma reflexão sobre as afirmações a seguir, seguida de uma discussão com base nos parágrafos escritos na sequência desta seção.

**Afirmações**

1. Códigos de linguagens diferentes são diferentes?
2. Códigos intermediários são diferentes?
3. Códigos de máquina são diferentes?

Para a primeira questão, pode-se observar códigos de linguagens diferentes com instruções equivalentes. Por exemplo, todos os códigos dos arquivos citados a seguir realizam a mesma ação, mas foram escritos em linguagens diferentes:

* [aula1.asm](https://drive.google.com/file/d/1Z-Oqbbg7Tm-4mZqGjAbgegVuo2cLKA8K/view?usp=drive_link)
* [aula1.c](https://drive.google.com/file/d/15LIuYOMxTMH3OKEojIVoIOdOmMtrbb87/view?usp=drive_link)
* [aula1.hs](https://drive.google.com/file/d/1_jZCW_q8dH_p8wa_jyo58PMtBqhkW3Hd/view?usp=drive_link)
* [aula1.java](https://drive.google.com/file/d/1VShF03yy6Fytf0WX6vc_kvyTfR_d7jdD/view?usp=drive_link)

Para a segunda questão, pode-se observar se os arquivos intermediários gerados por códigos escritos em linguagens diferentes também são diferentes. O comando abaixo gera o código intermediário do arquivo aula1.c:

**gcc -S aula1.c**

O comando acima gera um código Assembly (arquivo [aula1.s](https://drive.google.com/file/d/1AXuesImVUhkEs35B2cxuNKNGKLJ0Qn2F/view?usp=sharing)) para o código aula1.c, sendo uma representação intermediária dele. Ao comparar esse código com o código do arquivo aula1.asm, percebe-se que os arquivos são diferentes. A comparação também pode ser feita de forma automatizada com a ferramenta de linha de comando diff, como no comando a seguir:

**diff aula1.asm aula1.s**

Para a terceira questão, deve-se observar os códigos binários gerados. Códigos binários não são legíveis para humanos, então a análise pode ser feita diretamente pela ferramenta diff a partir da comparação entre os arquivos executáveis gerados para cada código.

O comando abaixo gera um executável chamado [aula1c](https://drive.google.com/file/d/1KCSQ-f0WnH2AxkYjZyWNbWmN8nA3Dfke/view?usp=drive_link), gerado a partir do código C do arquivo aula1.c:

**gcc aula1.c -o aula1c**

O comando abaixo gera um executável chamado [aula1asm](https://drive.google.com/file/d/1kNVP1ATAxEYVQOn2PDYAx-UDBOFkEjDu/view?usp=sharing), gerado a partir do código Assembly do arquivo aula1.asm:

**nasm -f elf64 aula1.asm; ld aula1.o -o aula1asm**

A comparação dos arquivos pode ser feita com o comando:

**diff aula1c aula1asm**

A partir desses executáveis é possível ainda desmontar os códigos, ou seja, a partir do código binário, buscar uma representação equivalente em linguagem de montagem. Isso pode ser feito com a ferramenta objdump, como nos exemplos a seguir.

O comando abaixo desmonta o arquivo aula1asm:

**objdump -S aula1asm**

A resposta obtida está exposta no quadro abaixo:

| **aula1asm: formato do arquivo elf64-x86-64**  **Desmontagem da seção .text:**  **00000000004000b0 <\_start>:**  **4000b0: ba 0e 00 00 00 mov $0xe,%edx**  **4000b5: b9 d0 00 60 00 mov $0x6000d0,%ecx**  **4000ba: bb 01 00 00 00 mov $0x1,%ebx**  **4000bf: b8 04 00 00 00 mov $0x4,%eax**  **4000c4: cd 80 int $0x80**  **4000c6: b8 01 00 00 00 mov $0x1,%eax**  **4000cb: cd 80 int $0x80** |
| --- |

O comando abaixo desmonta o arquivo aula1c:

**objdump -S aula1c**

A saída obtida está exposta no quadro abaixo:

| **aula1c: formato do arquivo elf64-x86-64**  **Desmontagem da seção .init:**  **00000000000004e8 <\_init>:**  **4e8: 48 83 ec 08 sub $0x8,%rsp**  **4ec: 48 8b 05 f5 0a 20 00 mov 0x200af5(%rip),%rax # 200fe8 <\_\_gmon\_start\_\_>**  **4f3: 48 85 c0 test %rax,%rax**  **4f6: 74 02 je 4fa <\_init+0x12>**  **4f8: ff d0 callq \*%rax**  **4fa: 48 83 c4 08 add $0x8,%rsp**  **4fe: c3 retq**  **Desmontagem da seção .plt:**  **0000000000000500 <.plt>:**  **500: ff 35 ba 0a 20 00 pushq 0x200aba(%rip) # 200fc0 <\_GLOBAL\_OFFSET\_TABLE\_+0x8>**  **506: ff 25 bc 0a 20 00 jmpq \*0x200abc(%rip) # 200fc8 <\_GLOBAL\_OFFSET\_TABLE\_+0x10>**  **50c: 0f 1f 40 00 nopl 0x0(%rax)**  **0000000000000510 <puts@plt>:**  **510: ff 25 ba 0a 20 00 jmpq \*0x200aba(%rip) # 200fd0 <puts@GLIBC\_2.2.5>**  **516: 68 00 00 00 00 pushq $0x0**  **51b: e9 e0 ff ff ff jmpq 500 <.plt>**  **Desmontagem da seção .plt.got:**  **0000000000000520 <\_\_cxa\_finalize@plt>:**  **520: ff 25 d2 0a 20 00 jmpq \*0x200ad2(%rip) # 200ff8 <\_\_cxa\_finalize@GLIBC\_2.2.5>**  **526: 66 90 xchg %ax,%ax**  **Desmontagem da seção .text:**  **0000000000000530 <\_start>:**  **530: 31 ed xor %ebp,%ebp**  **532: 49 89 d1 mov %rdx,%r9**  **535: 5e pop %rsi**  **536: 48 89 e2 mov %rsp,%rdx**  **539: 48 83 e4 f0 and $0xfffffffffffffff0,%rsp**  **53d: 50 push %rax**  **53e: 54 push %rsp**  **53f: 4c 8d 05 7a 01 00 00 lea 0x17a(%rip),%r8 # 6c0 <\_\_libc\_csu\_fini>**  **546: 48 8d 0d 03 01 00 00 lea 0x103(%rip),%rcx # 650 <\_\_libc\_csu\_init>**  **54d: 48 8d 3d e6 00 00 00 lea 0xe6(%rip),%rdi # 63a <main>**  **554: ff 15 86 0a 20 00 callq \*0x200a86(%rip) # 200fe0 <\_\_libc\_start\_main@GLIBC\_2.2.5>**  **55a: f4 hlt**  **55b: 0f 1f 44 00 00 nopl 0x0(%rax,%rax,1)**  **0000000000000560 <deregister\_tm\_clones>:**  **560: 48 8d 3d a9 0a 20 00 lea 0x200aa9(%rip),%rdi # 201010 <\_\_TMC\_END\_\_>**  **567: 55 push %rbp**  **568: 48 8d 05 a1 0a 20 00 lea 0x200aa1(%rip),%rax # 201010 <\_\_TMC\_END\_\_>**  **56f: 48 39 f8 cmp %rdi,%rax**  **572: 48 89 e5 mov %rsp,%rbp**  **575: 74 19 je 590 <deregister\_tm\_clones+0x30>**  **577: 48 8b 05 5a 0a 20 00 mov 0x200a5a(%rip),%rax # 200fd8 <\_ITM\_deregisterTMCloneTable>**  **57e: 48 85 c0 test %rax,%rax**  **581: 74 0d je 590 <deregister\_tm\_clones+0x30>**  **583: 5d pop %rbp**  **584: ff e0 jmpq \*%rax**  **586: 66 2e 0f 1f 84 00 00 nopw %cs:0x0(%rax,%rax,1)**  **58d: 00 00 00**  **590: 5d pop %rbp**  **591: c3 retq**  **592: 0f 1f 40 00 nopl 0x0(%rax)**  **596: 66 2e 0f 1f 84 00 00 nopw %cs:0x0(%rax,%rax,1)**  **59d: 00 00 00**  **00000000000005a0 <register\_tm\_clones>:**  **5a0: 48 8d 3d 69 0a 20 00 lea 0x200a69(%rip),%rdi # 201010 <\_\_TMC\_END\_\_>**  **5a7: 48 8d 35 62 0a 20 00 lea 0x200a62(%rip),%rsi # 201010 <\_\_TMC\_END\_\_>**  **5ae: 55 push %rbp**  **5af: 48 29 fe sub %rdi,%rsi**  **5b2: 48 89 e5 mov %rsp,%rbp**  **5b5: 48 c1 fe 03 sar $0x3,%rsi**  **5b9: 48 89 f0 mov %rsi,%rax**  **5bc: 48 c1 e8 3f shr $0x3f,%rax**  **5c0: 48 01 c6 add %rax,%rsi**  **5c3: 48 d1 fe sar %rsi**  **5c6: 74 18 je 5e0 <register\_tm\_clones+0x40>**  **5c8: 48 8b 05 21 0a 20 00 mov 0x200a21(%rip),%rax # 200ff0 <\_ITM\_registerTMCloneTable>**  **5cf: 48 85 c0 test %rax,%rax**  **5d2: 74 0c je 5e0 <register\_tm\_clones+0x40>**  **5d4: 5d pop %rbp**  **5d5: ff e0 jmpq \*%rax**  **5d7: 66 0f 1f 84 00 00 00 nopw 0x0(%rax,%rax,1)**  **5de: 00 00**  **5e0: 5d pop %rbp**  **5e1: c3 retq**  **5e2: 0f 1f 40 00 nopl 0x0(%rax)**  **5e6: 66 2e 0f 1f 84 00 00 nopw %cs:0x0(%rax,%rax,1)**  **5ed: 00 00 00**  **00000000000005f0 <\_\_do\_global\_dtors\_aux>:**  **5f0: 80 3d 19 0a 20 00 00 cmpb $0x0,0x200a19(%rip) # 201010 <\_\_TMC\_END\_\_>**  **5f7: 75 2f jne 628 <\_\_do\_global\_dtors\_aux+0x38>**  **5f9: 48 83 3d f7 09 20 00 cmpq $0x0,0x2009f7(%rip) # 200ff8 <\_\_cxa\_finalize@GLIBC\_2.2.5>**  **600: 00**  **601: 55 push %rbp**  **602: 48 89 e5 mov %rsp,%rbp**  **605: 74 0c je 613 <\_\_do\_global\_dtors\_aux+0x23>**  **607: 48 8b 3d fa 09 20 00 mov 0x2009fa(%rip),%rdi # 201008 <\_\_dso\_handle>**  **60e: e8 0d ff ff ff callq 520 <\_\_cxa\_finalize@plt>**  **613: e8 48 ff ff ff callq 560 <deregister\_tm\_clones>**  **618: c6 05 f1 09 20 00 01 movb $0x1,0x2009f1(%rip) # 201010 <\_\_TMC\_END\_\_>**  **61f: 5d pop %rbp**  **620: c3 retq**  **621: 0f 1f 80 00 00 00 00 nopl 0x0(%rax)**  **628: f3 c3 repz retq**  **62a: 66 0f 1f 44 00 00 nopw 0x0(%rax,%rax,1)**  **0000000000000630 <frame\_dummy>:**  **630: 55 push %rbp**  **631: 48 89 e5 mov %rsp,%rbp**  **634: 5d pop %rbp**  **635: e9 66 ff ff ff jmpq 5a0 <register\_tm\_clones>**  **000000000000063a <main>:**  **63a: 55 push %rbp**  **63b: 48 89 e5 mov %rsp,%rbp**  **63e: 48 8d 3d 8f 00 00 00 lea 0x8f(%rip),%rdi # 6d4 <\_IO\_stdin\_used+0x4>**  **645: e8 c6 fe ff ff callq 510 <puts@plt>**  **64a: 90 nop**  **64b: 5d pop %rbp**  **64c: c3 retq**  **64d: 0f 1f 00 nopl (%rax)**  **0000000000000650 <\_\_libc\_csu\_init>:**  **650: 41 57 push %r15**  **652: 41 56 push %r14**  **654: 49 89 d7 mov %rdx,%r15**  **657: 41 55 push %r13**  **659: 41 54 push %r12**  **65b: 4c 8d 25 56 07 20 00 lea 0x200756(%rip),%r12 # 200db8 <\_\_frame\_dummy\_init\_array\_entry>**  **662: 55 push %rbp**  **663: 48 8d 2d 56 07 20 00 lea 0x200756(%rip),%rbp # 200dc0 <\_\_init\_array\_end>**  **66a: 53 push %rbx**  **66b: 41 89 fd mov %edi,%r13d**  **66e: 49 89 f6 mov %rsi,%r14**  **671: 4c 29 e5 sub %r12,%rbp**  **674: 48 83 ec 08 sub $0x8,%rsp**  **678: 48 c1 fd 03 sar $0x3,%rbp**  **67c: e8 67 fe ff ff callq 4e8 <\_init>**  **681: 48 85 ed test %rbp,%rbp**  **684: 74 20 je 6a6 <\_\_libc\_csu\_init+0x56>**  **686: 31 db xor %ebx,%ebx**  **688: 0f 1f 84 00 00 00 00 nopl 0x0(%rax,%rax,1)**  **68f: 00**  **690: 4c 89 fa mov %r15,%rdx**  **693: 4c 89 f6 mov %r14,%rsi**  **696: 44 89 ef mov %r13d,%edi**  **699: 41 ff 14 dc callq \*(%r12,%rbx,8)**  **69d: 48 83 c3 01 add $0x1,%rbx**  **6a1: 48 39 dd cmp %rbx,%rbp**  **6a4: 75 ea jne 690 <\_\_libc\_csu\_init+0x40>**  **6a6: 48 83 c4 08 add $0x8,%rsp**  **6aa: 5b pop %rbx**  **6ab: 5d pop %rbp**  **6ac: 41 5c pop %r12**  **6ae: 41 5d pop %r13**  **6b0: 41 5e pop %r14**  **6b2: 41 5f pop %r15**  **6b4: c3 retq**  **6b5: 90 nop**  **6b6: 66 2e 0f 1f 84 00 00 nopw %cs:0x0(%rax,%rax,1)**  **6bd: 00 00 00**  **00000000000006c0 <\_\_libc\_csu\_fini>:**  **6c0: f3 c3 repz retq**  **Desmontagem da seção .fini:**  **00000000000006c4 <\_fini>:**  **6c4: 48 83 ec 08 sub $0x8,%rsp**  **6c8: 48 83 c4 08 add $0x8,%rsp**  **6cc: c3 retq** |
| --- |

A partir das ações realizadas é possível responder às questões da seção, fornecendo justificativas adequadas para cada caso.

## Conclusão

Na disciplina de Paradigmas de Programação, a complexidade e a diversidade das linguagens de programação e seus principais paradigmas serão estudados. O estudo dessa área contribui para a expressão de ideias e a criação de soluções computacionais a partir de uma visão holística sobre paradigmas. Desde o conceito fundamental de linguagem até a análise de diferentes paradigmas e suas aplicações, este material aborda uma visão geral e introdutória sobre linguagens de programação.

Compreender características, paradigmas e aplicações específicas de linguagens de programação torna o(a) leitor(a) capaz de tomar decisões informadas ao escolher a linguagem mais adequada para um determinado projeto, considerando fatores como facilidade de aprendizado, desempenho, comunidade de suporte e demanda no mundo de trabalho.

A discussão sobre as gerações de linguagens de programação traz uma reflexão sobre a evolução do campo da Computação. Desde linguagens de máquina até redes neurais e aprendizado de máquina, percebe-se que linguagens de programação se adaptaram e se especializaram para atender as necessidades de diferentes domínios de aplicação.

Na exploração das diferenças entre códigos escritos em linguagens distintas, intermediárias e de máquina, observa-se a computação em seu nível mais fundamental. A interpretação desses códigos revela a complexidade por trás da aparente simplicidade dos programas de computador, permitindo ao leitor refletir sobre como os algoritmos são traduzidos em instruções compreensíveis pelo hardware. Esse entendimento auxilia o programador a escrever códigos mais eficientes, robustos e adaptáveis.

## Tarefas

1. Realize seu [cadastro no Beecrowd](https://judge.beecrowd.com/pt/register).
2. Realize a leitura do capítulo 1 do livro *Concepts of programming languages*, de Robert W. Sebesta.

## Referências

OXFORD LANGUAGES. Linguagem. In: OXFORD LANGUAGES. 2024. Disponível em: <https://www.google.com/search?q=define%3Alinguagem>. Acesso em: 13 fev. 2024.

SEBESTA, Robert W. Concepts of programming languages. Pearson. 10th ed. 2012.

SEBESTA, Robert W. Concepts of programming languages. Bookman. 9ª ed. 2011.

TIOBE. TIOBE Index for February 2024. Disponível em: <https://www.tiobe.com/tiobe-index/>. Acesso em: 14 fev. 2024.