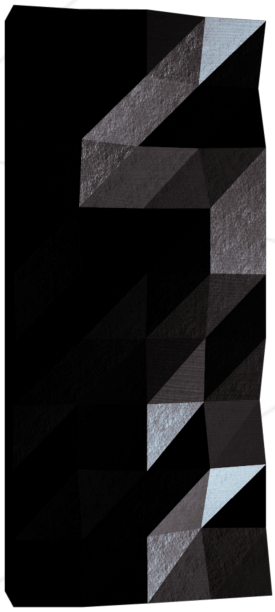

Philosophers

Nunca pensei que um filósofo seria tão mortal

Preâmbulo:
Neste projeto, você aprenderá os conceitos básicos de encadeamento de um processo.
Você aprenderá como criar threads e explorar o uso de mutexes.

Versão: 12.0

Sumário

Capítulo I

Introduction

Filosofia (do grego, *philosophia*, literalmente "amor à sabedoria") é é o estudo de questões gerais e fundamentais sobre a existência, conhecimento, valores, razão, mente e linguagem. Essas questões são frequentemente formuladas como problemas para serem analisados ou resolvidos. O termo provavelmente foi cunhado por Pitágoras (c. 570 – 495 A.C.). Os métodos filosóficos incluem questionamento, discussão crítica, argumentação racional e apresentação sistemática.

Questões filosóficas clássicas incluem: Qualquer coisa pode ser conhecida e provada? O que é mais real? Filósofos também abordam questões mais práticas e concretas como: Existe uma maneira melhor de se viver? É melhor ser justo ou injusto (se uma pessoa pode se safar disso)? Os humanos têm livre arbítrio?

Historicamente, o termos "filosofia"se referia a qualquer tipo de conhecimento. Da época do filósofo da Grécia Antiga Aristóteles até o século XIX, a "filosofia natural"abrengia astronomia, medicina e física. Por exemplo, o trabalho de Newton de 1687, *Princípios Matemáticos da Filosofia Natural*, foi classificado mais tarde como um livro de física.

No século XIX, o crescimento de universidades de pesquisa modernas levou a filosofia acadêmica e outras disciplinas a se profissionalizarem e se especializarem. Na era moderna, algumas investigações que tradicionalmente eram parte da filosofia se tornaram disciplinas acadêmicas, incluindo psicologia, sociologia,, linguística e economia.

Outras investigações mais relacionadas à arte, ciência, polígica, ou outras atividades permaneceram parte da filosofia. Por exemplo, a beleza é objetiva ou subjetiva? Há muitos métodos científicos ou somente um? A utopia política é um sonho esperançoso ou uma fantasia sem esperanças? Os principais subcampos da filosofia acadêmica incluem metafísica ("concernente à natureza fundamental da realidade e do ser"), epistemologia (sobre a "natureza e os fundamentos do conhecimento [e]... seus limites e validade"), ética, estética, filosofia política, lógica e filosofia da ciência.

Capítulo II

Instruções Comuns

- Seu projeto deve ser escrito em C.
- Seu projeto deve ser escrito de acordo com a Norma. Se você tiver arquivos/funções bônus, eles serão incluídos na verificação da norma, e você receberá uma nota `0` se houver algum erro de norma.
- Suas funções não devem sair inesperadamente (segmentation fault, bus error, double free, etc.) exceto por comportamento indefinido. Se isso ocorrer, seu projeto será considerado não funcional e receberá uma nota `0` durante a avaliação.
- Toda memória alocada na heap deve ser liberada corretamente quando necessário. Vazamentos de memória não serão tolerados.
- Se o enunciado exigir, você deve submeter um `Makefile` que compile seus arquivos fonte para a saída requerida com as flags `-Wall`, `-Wextra`, e `-Werror`, usando `cc`. Adicionalmente, seu `Makefile` não deve realizar re-links desnecessários.
- Seu `Makefile` deve conter pelo menos as regras `$(NAME)`, `all`, `clean`, `fclean` e `re`.
- Para submeter bônus para seu projeto, você deve incluir uma regra `bonus` em seu `Makefile`, que adicionará todos os diversos headers, bibliotecas, ou funções que não são permitidas na parte principal do projeto. Os bônus devem ser colocados em arquivos `_bonus.{c/h}`, a menos que o enunciado especifique o contrário. A avaliação das partes obrigatórias e bônus é conduzida separadamente.
- Se seu projeto permitir que você use sua `libft`, você deve copiar suas fontes e seu `Makefile` associado para uma pasta `libft`. O `Makefile` do seu projeto deve compilar a biblioteca usando seu `Makefile`, e então compilar o projeto.
- Nós encorajamos você a criar programas de teste para seu projeto, mesmo que este trabalho **não precise ser submetido e não será avaliado**. Isso lhe dará a oportunidade de testar facilmente seu trabalho e o trabalho de seus colegas. Você achará esses testes especialmente úteis durante sua defesa. De fato, durante a defesa, você está livre para usar seus testes e/ou os testes do colega que você está avaliando.

- Submeta seu trabalho para o repositório Git designado. Somente o trabalho no repositório Git será avaliado. Se o Deepthought for designado para avaliar seu trabalho, isso ocorrerá após as avaliações de seus colegas. Se um erro acontecer em qualquer seção do seu trabalho durante a avaliação do Deepthought, a avaliação será interrompida.

Capítulo III

Instruções para IA

Contexto

Durante sua jornada de aprendizado, a IA pode auxiliar em diversas tarefas. Dedique tempo para explorar as várias capacidades das ferramentas de IA e como elas podem apoiar seu trabalho. No entanto, sempre as utilize com cautela e avalie criticamente os resultados. Seja código, documentação, ideias ou explicações técnicas, você nunca pode ter certeza absoluta de que sua pergunta foi bem formulada ou que o conteúdo gerado é preciso. Seus colegas são um recurso valioso para ajudá-lo a evitar erros e pontos cegos.

Mensagem principal

- Use a IA para reduzir tarefas repetitivas ou tediosas.
- Desenvolva habilidades de prompt — tanto para codificação quanto para outras tarefas — que beneficiarão sua carreira futura.
- Aprenda como os sistemas de IA funcionam para melhor antecipar e evitar riscos comuns, vieses e questões éticas.
- Continue desenvolvendo habilidades técnicas e interpessoais trabalhando com seus colegas.
- Use apenas conteúdo gerado por IA que você entenda completamente e pelo qual possa se responsabilizar.

Regras para o aluno:

- Você deve dedicar tempo para explorar as ferramentas de IA e entender como elas funcionam, para que possa usá-las eticamente e reduzir potenciais vieses.
- Você deve refletir sobre seu problema antes de criar o prompt — isso ajuda você a escrever prompts mais claros, detalhados e relevantes, usando vocabulário preciso.

- Você deve desenvolver o hábito de verificar, revisar, questionar e testar sistematicamente tudo o que for gerado por IA.
- Você deve sempre buscar revisão por pares — não confie apenas em sua própria validação.

Resultados da fase:

- Desenvolver habilidades de prompt tanto para uso geral quanto para áreas específicas.
- Aumentar sua produtividade com o uso eficaz de ferramentas de IA.
- Continuar fortalecendo o pensamento computacional, resolução de problemas, adaptabilidade e colaboração.

Comentários e exemplos:

- Você encontrará regularmente situações — exames, avaliações e mais — onde você deve demonstrar compreensão real. Esteja preparado, continue desenvolvendo suas habilidades técnicas e interpessoais.
- Explicar seu raciocínio e debater com colegas frequentemente revela lacunas em sua compreensão. Priorize a aprendizagem colaborativa.
- As ferramentas de IA geralmente carecem do seu contexto específico e tendem a fornecer respostas genéricas. Seus colegas, que compartilham seu ambiente, podem oferecer insights mais relevantes e precisos.
- Onde a IA tende a gerar a resposta mais provável, seus colegas podem fornecer perspectivas alternativas e nuances valiosas. Conte com eles como um ponto de verificação de qualidade.

✓ Boa prática:

> Pergunto à IA: “Como testo uma função de ordenação?” Ela me dá algumas ideias. Eu as testo e reviso os resultados com um colega. Nós refinamos a abordagem juntos.

✗ Má prática:

> Peço à IA para escrever uma função inteira, copio e colo no meu projeto. Durante a avaliação por pares, não consigo explicar o que ela faz ou porquê. Perco credibilidade — e reprovo no meu projeto.

✓ Boa prática:

Uso a IA para ajudar a projetar um analisador sintático. Então, analiso a lógica com um colega. Detectamos dois erros e reescrevemos juntos — melhor, mais limpo e totalmente compreendido.

✗ Má prática:

Deixo o Copilot gerar meu código para uma parte importante do meu projeto. Ele compila, mas não consigo explicar como ele lida com pipes. Durante a avaliação, não consigo justificar e reprovo no meu projeto.

Capítulo IV

Visão geral

Aqui estão as principais coisas que você precisa saber para ter sucesso neste projeto:

- Um ou mais filósofos sentam numa mesa redonda.
  Há uma travessa grande de espaguete no meio da mesa.

- Os filósofos se revesam ao comer, pensar e dormir.
  Enquanto eles estão comendo, eles não estão pensando nem dormindo;
  enquanto pensam, eles não estão comendo nem dormindo;
  e, claro, enquanto estão dormindo, eles não estão comendo nem pensando.

- Há também garfos na mesa. Há **a mesma quantidade de garfos que a de filósofos**.

- Como comer espaguete com somente um garfo não é muito prático, um filósofo deve pegar tanto o garfo à sua direita quanto o garfo à sua esquerda antes de comer.

- Quando um filósofo termina de comer, ele coloca seus garfos de volta na na mesa e começa a dormir. Uma vez acordado, ele começa a pensar novamente. A simulação para quando um filósofo morre de fome.

- Todo filósofo precisa comer e nunca deve passar fome.

- Os filósofos não se comunicam uns com os outros.

- Os filósofos não sabem se outro filósofo está prestes a morrer.

- Nem é preciso dizer que os filósofos devem evitar morrer!

Capítulo V

Regras gerais

Você precisa escrever um programa para a parte obrigatória e outro para a parte bônus (se você decidir fazer a parte bônus). Você precisa cumprir com as seguintes regras:

- Variáveis globais são proibidas!
- Seu(s) programa(s) deve(m) receber os seguintes argumentos:
  `number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]`
  - `number_of_philosophers`: O número de filósofos e também o número de garfos.
  - `time_to_die` (em milisegundos): Se um filósofo não tiver começado a comer dentro de `time_to_die` milisegundos desde o início da sua última refeição ou o início da simulação, ele morre.
  - `time_to_eat` (em milisegundos): O tempo que um filósofo leva para comer. Durante esse tempo, ele precisará segurar dois garfos.
  - `time_to_sleep` (em milisegundos): O tempo que um filósofo passará dormindo.
  - `number_of_times_each_philosopher_must_eat` (argumento opcional): Se todos os filósofo comeram pelo menos `number_of_times_each_philosopher_must_eat` vezes, a simulação para. Se não for especificado, a simulação para quando um filósofo morre.
- Cada filósofo tem um número que vai de `1` a `number_of_philosophers`.
- O filósofo número `1` senta ao lado do filósofo `number_of_philosophers`. Qualquer outro filósofo, de número `N`, senta entre os filósofos `N` - 1 e o filósofo `N` + 1.

Sobre os logs do seu programa:

- Qualquer mudança de estado de um filósofo deve ser registrada da seguinte forma:
  - `timestamp_in_ms X has taken a fork`
  - `timestamp_in_ms X is eating`
  - `timestamp_in_ms X is sleeping`
  - `timestamp_in_ms X is thinking`
  - `timestamp_in_ms X died`

  Substitua `timestamp_in_ms` *por o timestamp atual em milisegundos e* `X` *pelo número do filósofo.*

- Uma mensagem de mudança de estado que foi mostrada não pode se sobrepor a outra mensagem.
- Uma mensagem anunciando a morte de um filósofo deve ser mostrada em até 10 ms da sua verdadeira morte.
- De novo, os filósofos devem evitar morrer!

Seu programa não deve ter **data races.**

Capítulo VI

Mandatory part

Nome do programa	`philo`
Arquivos para entregar	`Makefile, *.h, *.c, no diretório philo/`
Makefile	`NAME, all, clean, fclean, re`
Argumentos	`number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]`
Funções externas autorizadas	`memset, printf, malloc, free, write, usleep, gettimeofday, pthread_create, pthread_detach, pthread_join, pthread_mutex_init, pthread_mutex_destroy, pthread_mutex_lock, pthread_mutex_unlock`
Libft autorizada	`Não`
Descrição	`Filósofos com threads e mutexes`

As regras específicas para a parte obrigatória são:

- Cada filósofo deve ser representado como uma thread separada.
- Há um garfo entre cada par de filósofos. Portanto, se há vários filósofos, cada filósofo tem um garfo em seu lado esquerdo e um garfo em seu lado direito. Se há somente um filósofo, ele terá acesso a somente um garfo.
- Para previnir que os filósofos tenham garfos duplicados, você deve proteger cada estado de garfo com um mutex.

Capítulo VII

Bonus part

Nome do programa	`philo_bonus`
Arquivos para entregar	`Makefile, *.h, *.c, no diretório philo_bonus/`
Makefile	`NAME, all, clean, fclean, re`
Argumentos	`number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]`
Funções externas autorizadas	`memset, printf, malloc, free, write, fork, kill, exit, pthread_create, pthread_detach, pthread_join, usleep, gettimeofday, waitpid, sem_open, sem_close, sem_post, sem_wait, sem_unlink`
Libft autorizada	`Não`
Descrição	`Filósofos com processos e semáforos`

O programa da parte bônus recebe os mesmos argumentos que o programa da parte obrigatória. Ele deve cumprir os requisitos do capítulo *Regras gerais*.

As regras específicas para a parte bônus são:

- Todos os garfos são colocados no meio da mesa.
- Eles não têm estados na memória, mas o número de garfos disponíveis é representado por um semáforo.
- Cada filósofo deve ser representado por um processo em separado. No entanto, o processo principal não deve agir como um filósofo.

`A parte bônus só será avaliada se a parte obrigatória estiver PERFEITA. Perfeita significa que a parte obrigatória foi integralmente feita e funciona sem apresentar falhas. Se você não tiver passado TODOS os requisitos obrigatórios, sua parte bônus não será avaliada.`

Capítulo VIII

Submission and peer-evaluation

Envie sua tarefa em seu repositório `Git` como de costume. Apenas o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar duas vezes os nomes de seus arquivos para garantir que estejam corretos.

Diretório da parte obrigatória: `philo/`

Diretório da parte bônus: `philo_bonus/`

Durante a avaliação, uma breve **modificação do projeto** pode ser ocasionalmente solicitada. Isso pode envolver uma pequena mudança de comportamento, algumas linhas de código para escrever ou reescrever, ou um recurso fácil de adicionar.

Embora esta etapa possa **não ser aplicável a todos os projetos**, você deve estar preparado para ela se for mencionada nas diretrizes de avaliação.

Esta etapa visa verificar sua compreensão real de uma parte específica do projeto. A modificação pode ser realizada em qualquer ambiente de desenvolvimento que você escolher (por exemplo, sua configuração usual), e deve ser viável em poucos minutos — a menos que um prazo específico seja definido como parte da avaliação.
Você pode, por exemplo, ser solicitado a fazer uma pequena atualização em uma função ou script, modificar uma exibição ou ajustar uma estrutura de dados para armazenar novas informações, etc.

Os detalhes (escopo, alvo, etc.) serão especificados nas **diretrizes de avaliação** e podem

variar de uma avaliação para outra para o mesmo projeto.

```
V              L      (     L   p     aW    8
 ) *   q fx  2          u  L   5  kt  6   ::/ .
?     p  x  >g    0 Jb    n    R     V
       7                        vo
                              v   c    p t     g   j

           S        4      H            .
                    I      6         <  )
          A2   5  )     E       ?            d
   B        f         -         Z         ‘
                                              * v
 ,           ;  I       I
s   C   :
```