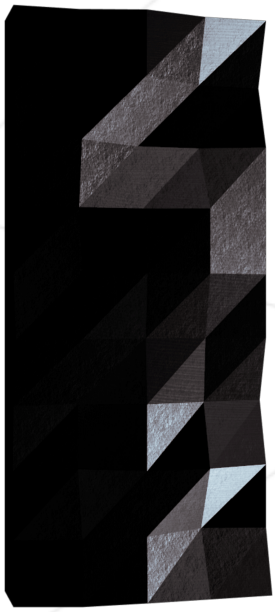

Philosophers

I never thought philosophy would be so deadly

Summary:
In this project, you will learn the basics of threading a process.
You will learn how to create threads and explore the use of mutexes.

Version: 12.0

Contents

Chapter I

Introduction

Philosophy (from Greek, *philosophia*, literally "love of wisdom") is the study of general and fundamental questions about existence, knowledge, values, reason, mind, and language. Such questions are often framed as problems to be analyzed or solved. The term was probably coined by Pythagoras (c. 570 – 495 BCE). Philosophical methods include questioning, critical discussion, rational argument, and systematic presentation.

Classic philosophical questions include: Can anything be truly known and proven? What is most real? Philosophers also pose more practical and concrete questions such as: Is there the best way to live? Is it better to be just or unjust (if one can get away with it)? Do humans have free will?

Historically, the term 'philosophy' referred to any body of knowledge. From the time of Ancient Greek philosopher Aristotle to the 19th century, "natural philosophy" encompassed astronomy, medicine, and physics. For example, Newton's 1687 work, *Mathematical Principles of Natural Philosophy*, was later classified as a physics book.

In the 19th century, the growth of modern research universities led academic philosophy and other disciplines to professionalize and specialize. In the modern era, some investigations that were traditionally part of philosophy became separate academic disciplines, including psychology, sociology, linguistics, and economics.

Other investigations closely related to art, science, politics, or other pursuits remained part of philosophy. For example, is beauty objective or subjective? Are there many scientific methods or just one? Is political utopia a hopeful dream or hopeless fantasy? Major sub-fields of academic philosophy include metaphysics ("concerned with the fundamental nature of reality and being"), epistemology (about the "nature and grounds of knowledge [and]... its limits and validity"), ethics, aesthetics, political philosophy, logic and philosophy of science.

Chapter II

Common Instructions

- Your project must be written in C.
- Your project must be written in accordance with the Norm. If you have bonus files/functions, they are included in the norm check, and you will receive a `0` if there is a norm error.
- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc.) except for undefined behavior. If this occurs, your project will be considered non-functional and will receive a `0` during the evaluation.
- All heap-allocated memory must be properly freed when necessary. Memory leaks will not be tolerated.
- If the subject requires it, you must submit a `Makefile` that compiles your source files to the required output with the flags `-Wall`, `-Wextra`, and `-Werror`, using `cc`. Additionally, your `Makefile` must not perform unnecessary relinking.
- Your `Makefile` must contain at least the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- To submit bonuses for your project, you must include a `bonus` rule in your `Makefile`, which will add all the various headers, libraries, or functions that are not allowed in the main part of the project. Bonuses must be placed in `_bonus.{c/h}` files, unless the subject specifies otherwise. The evaluation of mandatory and bonus parts is conducted separately.
- If your project allows you to use your `libft`, you must copy its sources and its associated `Makefile` into a `libft` folder. Your project's `Makefile` must compile the library by using its `Makefile`, then compile the project.
- We encourage you to create test programs for your project, even though this work **does not need to be submitted and will not be graded**. It will give you an opportunity to easily test your work and your peers' work. You will find these tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to the assigned Git repository. Only the work in the Git repository will be graded. If Deepthought is assigned to grade your work, it will occur

after your peer-evaluations. If an error happens in any section of your work during Deepthought's grading, the evaluation will stop.

Chapter III

AI Instructions

Context

During your learning journey, AI can assist with many different tasks. Take the time to explore the various capabilities of AI tools and how they can support your work. However, always approach them with caution and critically assess the results. Whether it's code, documentation, ideas, or technical explanations, you can never be completely sure that your question was well-formed or that the generated content is accurate. Your peers are a valuable resource to help you avoid mistakes and blind spots.

Main message

- Use AI to reduce repetitive or tedious tasks.
- Develop prompting skills — both coding and non-coding — that will benefit your future career.
- Learn how AI systems work to better anticipate and avoid common risks, biases, and ethical issues.
- Continue building both technical and power skills by working with your peers.
- Only use AI-generated content that you fully understand and can take responsibility for.

Learner rules:

- You should take the time to explore AI tools and understand how they work, so you can use them ethically and reduce potential biases.
- You should reflect on your problem before prompting — this helps you write clearer, more detailed, and more relevant prompts using accurate vocabulary.
- You should develop the habit of systematically checking, reviewing, questioning, and testing anything generated by AI.
- You should always seek peer review — don't rely solely on your own validation.

Phase outcomes:

- Develop both general-purpose and domain-specific prompting skills.
- Boost your productivity with effective use of AI tools.
- Continue strengthening computational thinking, problem-solving, adaptability, and collaboration.

Comments and examples:

- You'll regularly encounter situations — exams, evaluations, and more — where you must demonstrate real understanding. Be prepared, keep building both your technical and interpersonal skills.
- Explaining your reasoning and debating with peers often reveals gaps in your understanding. Make peer learning a priority.
- AI tools often lack your specific context and tend to provide generic responses. Your peers, who share your environment, can offer more relevant and accurate insights.
- Where AI tends to generate the most likely answer, your peers can provide alternative perspectives and valuable nuance. Rely on them as a quality checkpoint.

✓ Good practice:

> I ask AI: "How do I test a sorting function?" It gives me a few ideas. I try them out and review the results with a peer. We refine the approach together.

✗ Bad practice:

> I ask AI to write a whole function, copy-paste it into my project. During peer-evaluation, I can't explain what it does or why. I lose credibility — and I fail my project.

✓ Good practice:

> I use AI to help design a parser. Then I walk through the logic with a peer. We catch two bugs and rewrite it together — better, cleaner, and fully understood.

✗ Bad practice:

> I let Copilot generate my code for a key part of my project. It compiles, but I can't explain how it handles pipes. During the evaluation, I fail to justify and I fail my project.

Chapter IV

Overview

Here are the key things you need to know to succeed in this assignment:

- One or more philosophers sit at a round table.
  There is a large bowl of spaghetti in the middle of the table.
- The philosophers take turns eating, thinking, and sleeping.
  While they are eating, they are not thinking nor sleeping;
  while thinking, they are not eating nor sleeping;
  and, of course, while sleeping, they are not eating nor thinking.
- There are also forks on the table. There are **as many forks as philosophers**.
- Since eating spaghetti with just one fork is impractical, a philosopher must pick up both the fork to their right and the fork to their left before eating.
- When a philosopher has finished eating, they put their forks back on the table and start sleeping. Once awake, they start thinking again. The simulation stops when a philosopher dies of starvation.
- Every philosopher needs to eat and should never starve.
- Philosophers do not communicate with each other.
- Philosophers do not know if another philosopher is about to die.
- Needless to say, philosophers should avoid dying!

Chapter V

Global rules

You have to write a program for the mandatory part and another one for the bonus part (if you decide to do the bonus part). They both have to comply with the following rules:

- Global variables are forbidden!
- Your program(s) must take the following arguments:
  `number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]`
  - `number_of_philosophers`: The number of philosophers and also the number of forks.
  - `time_to_die` (in milliseconds): If a philosopher has not started eating within `time_to_die` milliseconds since the start of their last meal or the start of the simulation, they die.
  - `time_to_eat` (in milliseconds): The time it takes for a philosopher to eat. During that time, they will need to hold two forks.
  - `time_to_sleep` (in milliseconds): The time a philosopher will spend sleeping.
  - `number_of_times_each_philosopher_must_eat` (optional argument): If all philosophers have eaten at least `number_of_times_each_philosopher_must_eat` times, the simulation stops. If not specified, the simulation stops when a philosopher dies.
- Each philosopher has a number ranging from `1` to `number_of_philosophers`.
- Philosopher number `1` sits next to philosopher number `number_of_philosophers`. Any other philosopher, numbered `N`, sits between philosopher `N` - 1 and philosopher `N` + 1.

About the logs of your program:

- Any state change of a philosopher must be formatted as follows:
  - `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`

  Replace `timestamp_in_ms` *with the current timestamp in milliseconds and* `X` *with the philosopher number.*
- A displayed state message should not overlap with another message.
- A message announcing a philosopher's death must be displayed within 10 ms of their actual death.
- Again, philosophers should avoid dying!

`Your program must not have any` **`data races.`**

Chapter VI

Mandatory part

Program Name	`philo`
Files to Submit	`Makefile, *.h, *.c, in directory philo/`
Makefile	`NAME, all, clean, fclean, re`
Arguments	`number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]`
External Function	`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 authorized	`No`
Description	`Philosophers with threads and mutexes`

The specific rules for the mandatory part are:

- Each philosopher must be represented as a separate thread.
- There is one fork between each pair of philosophers. Therefore, if there are several philosophers, each philosopher has a fork on their left side and a fork on their right side. If there is only one philosopher, they will have access to just one fork.
- To prevent philosophers from duplicating forks, you should protect each fork's state with a mutex.

Chapter VII

Bonus part

Program Name	`philo_bonus`
Files to Submit	`Makefile, *.h, *.c, in directory philo_bonus/`
Makefile	`NAME, all, clean, fclean, re`
Arguments	`number_of_philosophers time_to_die time_to_eat time_to_sleep [number_of_times_each_philosopher_must_eat]`
External Function	`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 authorized	`No`
Description	`Philosophers with processes and semaphores`

The program of the bonus part takes the same arguments as the mandatory program. It has to comply with the requirements of the *Global rules* chapter.

The specific rules for the bonus part are:

- All the forks are put in the middle of the table.
- They have no states in memory, but the number of available forks is represented by a semaphore.
- Each philosopher must be represented as a separate process. However, the main process should not act as a philosopher.

`The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.`

Chapter VIII

Submission and peer-evaluation

Submit your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double-check the names of your files to ensure they are correct.

Mandatory part directory: `philo/`

Bonus part directory: `philo_bonus/`

During the evaluation, a brief **modification of the project** may occasionally be requested. This could involve a minor behavior change, a few lines of code to write or rewrite, or an easy-to-add feature.

While this step may **not be applicable to every project**, you must be prepared for it if it is mentioned in the evaluation guidelines.

This step is meant to verify your actual understanding of a specific part of the project. The modification can be performed in any development environment you choose (e.g., your usual setup), and it should be feasible within a few minutes — unless a specific timeframe is defined as part of the evaluation.
You can, for example, be asked to make a small update to a function or script, modify a display, or adjust a data structure to store new information, etc.

The details (scope, target, etc.) will be specified in the **evaluation guidelines** and may

vary from one evaluation to another for the same project.

```
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   :
```