DRAFT:
If there are unclear elements let me know.

Main assignment (1st chance)

A slice 4 my life!

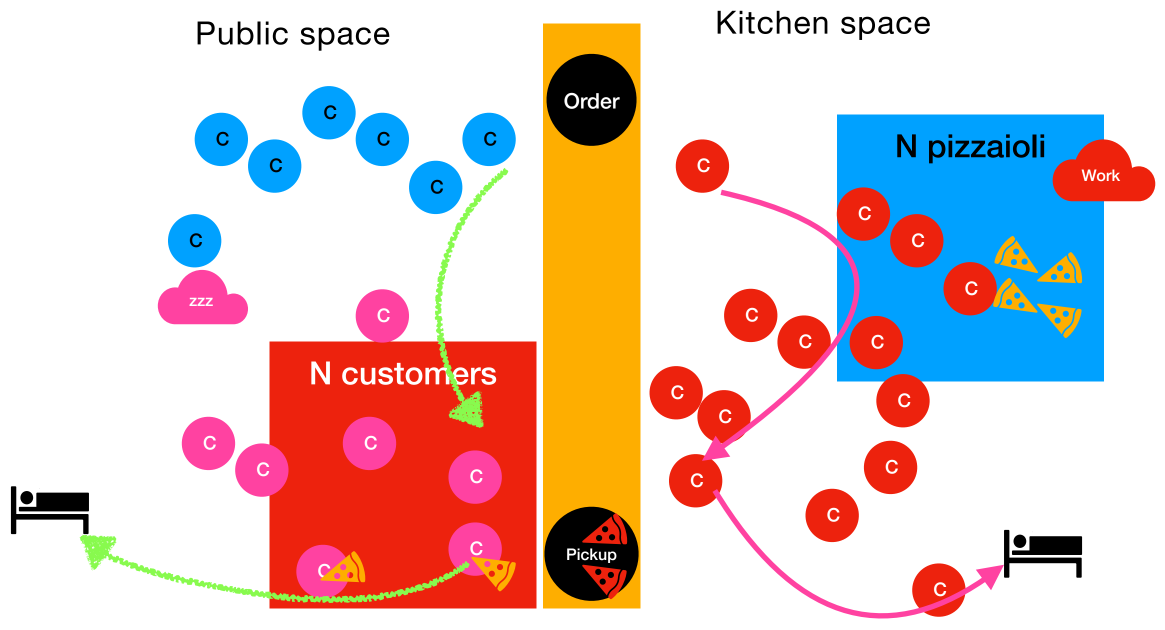
The setting:

We are setting up a fast pickup restaurant! Not a new thing. The food is an amazing kind of pizza! But there is only one dish served. You order it, wait, and get the first one you can. You can take one (ONLY ONE) slice from a dish and go! Don't worry about the others, they will not complain, they will not even notice it! But don't get tangled in others' slices! The space is small inside and everyone needs to take turns in getting the slice!

This restaurant has two locations where customers and pizzaioli can exchange pizza and orders. The first one is where you place your order, the second one is where you pick it up.

After a customer places the order, he will wait a certain amount of time and then go to the pickup location to get the order.

The pizzaioli are behind the counter and furiously preparing the same dish repeatedly. It is a pizza that takes some time but to avoid boredom for the cooks, every cook only cooks *one single slice every run*. They complete a dish (a pizza) making one slice each, and work together but the space is small! After they are done with a single order, they can go home and rest.



The code:

The provided code might or might not work. It is not implemented concurrently. Your duty as a developer is to complete correctly the given application. The application should become **multithreaded** reaching the maximum level of concurrency, so everything that can be done in a concurrent setting should be implemented that way.

This **concurrent** application (command line only) requires no interaction with the user. Any interaction with the user is forbidden.

You must grant the property of **fairness** and **correctness** in your assignment.

Every part of your code must be completed so no random interleaving can alter the outcome. So, it must be **thread-safe**.

If the strategies to protect your memory are excessive (read: overlocking, or locking things that should not or do not need locks), they will be evaluated as invalid. Remember that it is important to minimise the coverage of your locking (or number of semaphores). *Spinning* is forbidden.

The desired outcome is:

- Each **pizzaiolo**[1] gets one order (only one slice) from the order location and works with other “n” pizzaioli to make it “full”. Only the last pizzaiolo to work on it will deliver the full pizza to the pickup location and terminate. The other pizzaioli will just add the slice to the working surface and terminate. The pizza is considered done only if the dish contains “n” pizza slices. But each slice has its state (already done in code).

- Each **customer** will independently generate an order at the order location, take a small sleep, and pick up the first available slice of pizza at the pickup location. After that, it will terminate. The customers will get one and only one pizza slice produced by the cooks. The pizza slices are accessible only to “n” customers at a time (meaning no more than “n” customers can have access to a dish where the slices are located per time).
  - The order placed by each customer is of a single slice.
  - The pizza dish contains always “n” slices, each made by a different pizzaiolo.
  - The customers will take out the dish once it is empty.

[1] Pizzaiolo: is the Italian name for who makes a pizza, the plural in Italian is pizzaioli.

A precondition for the assignment is that the number of customers and pizzaioli is always the same at the beginning and an exact multiple of the n_slices (see code comments). There will be no testing of what is happening when the number is not a multiple of n_slices, but it is important to understand the consequences. So each cook and customer will just run once and never be reused. This is a simplification for handling the concurrent work.

There is no need to check for speed of execution. But take care that each task gets executed by who should do it.

Breaks ("thread.sleep") are there for specific purposes and should not be removed or moved from their logical spot, there is no need to add more either. Similarly, printouts are there for testing and logical purposes. *Do not remove them or alter them* in the **final** delivery. While testing, feel free to debug it as you prefer, even adding I/O operations. Just remember to clean up your code before the final delivery. I left for your convenience some commented printout. Remember to put them back as they were before delivery.

The main method will start threads in phases (already indicated in the method), feel free to change the content of the functions call as needed.

You can create new classes if needed.

You can add variables if needed.

You can alter the signatures of some of the methods (where allowed) if needed.

We highly encourage testing: rise and/or diminish the number of threads/dishes/slices/tasks to be completed that should be prepared. Remember that testing over different memory conditions (over extra load/with less load) can facilitate the evidence of some errors or abnormal behaviours.

The provided code has comments to guide you and help you address where to edit and where not to edit. Try to add as less changes as needed. Respect the structure and be considerate of what the comments are saying.

Note: the lists of orders and pickups at the end must always be zero. Every order must be ordered and picked up, if there are leftovers (or you are erasing traces of leftovers, it will be evaluated negatively).

Note: the code might result in much more elegant and simple by using this release overload—> https://learn.microsoft.com/en-us/dotnet/api/system.threading.semaphore.release?view=net-6.0#system-threading-semaphore-release(system-int32) It is not mandatory, but this makes things easier.

The delivery:

Every delivery group is made of **up to 2 students.** Only one student must submit the file(s). Retakers can choose anyone, no approval is needed.

The names on the code will represent who is part of the group. Your code MUST include the names and student numbers of each component.

The delivery will be in Microsoft Forms. Late submissions, email submissions, and chat submissions will be discarded and graded as ND.

Deliveries should NOT contain the **bin/obj** directory (or other temp files of your IDE).

LINK: https://forms.office.com/e/4TPjqqUu65
(copy and paste it, as occasionally it does not copy correctly).

Deadline: **January 24, 2025 at 11:30 pm**.

Changes to the submission are possible in Microsoft Forms, due to technical issues, any change after the deadline might invalidate the submission. So, do not make changes to the submission after the deadline.

When asked, at the end of the submission form, SAVE the link in your forms to edit the submission from there.

Only one submission (editable) per student is allowed.

FOLLOW THE INSTRUCTIONS ON THE SUBMISSION TO UPLOAD THE FILE.

The file ".zip" must be renamed to ".dot" to workaround the limitations of the form.

The testing and evaluation:

If your assignment **fails to meet the following points,** your grade might get a **negative assessment** without further testing.

- You do NOT require user interaction.
- Your code compiles correctly and does NOT require versions of ".Net" other than ".Net 6.x".

- Your code does NOT show any syntax errors.
- Your code does NOT crash.
- You did not alter the existing printout or sleep.
- You did not add in the delivery any printout or sleep.
- Your code does not alter the logic/algorithm of the assignment (change order of existing code).
- You are NOT using an auto-locking data structure or libraries (concurrent queues, blocking data structures, other data structures, or primitives that handle threads automatically, ex.: task library or “promises-like” functions of different kinds).
- You are NOT using parallel-for or other facilitating in-language control systems that avoid using threads in a raw context. Async-await structures are part of this category and thus: NOT allowed.
- You are NOT using spinning to synchronise threads (avoid spinning).
- You are NOT using external libraries that are not the standard Threads/ mutex/locks/semaphores.
- Your code is cross-platform (the solution will NOT run only on your system).
- You understand every single part of your code and which are the implications (to keep in mind: according to the course description an oral check is possible at any time to clarify your contribution).
- Your code is original (to keep in mind: according to the course description an oral check is possible at any time to clarify your contribution).
- Your delivery does NOT contain *bin* and *obj* directories or other temporary files.
- Your code implements what is requested.
- Your code does NOT implement some kind of a sequential version of the task provided.
- The delivery format is as requested (file naming, compression method for the solution, etc…).
- Your code is thread-safe.

Using any code-generating service is forbidden (**ChatGPT**-like or other autogenerating systems). Keep in mind that platforms of that kind tend to give always similar results to each other.

Code linter/autocomplete—> ok **Copilot**-like—>**forbidden**.

What will be tested?

Besides the previous requirements and minimal decency of code quality, we will check that **all the shared memory is correctly protected** and that the algorithm that synchronises the various tasks is carried out correctly by who should do it (and not altered). Your code must grant fairness and correctness in the application (for fairness it is delegated to the OS, assumed fair). We will check if the **protection** applied is sufficient and **only where necessary**, and evaluate negatively what is overprotected.

How will it be tested?

Partially automatically and partially by hand. The “automatic” testing will be on compiler errors/warnings, formatting of the files, and printout results (in case the reviewer needs to, will also run these tests by hand). The “by hand” part will be carried by command line calls as “dotnet run” in the project directory. Are you using Visual Studio? Consider testing by running the code via the command line interface. Everything should be OK with the same setting as the provided project.

Common questions:

Do you doubt a concept relevant to your code? You can ask in the forum. Please, refrain from asking questions (in public or private) as “Is this ok?” or “Is it sufficient like this?”. Teachers are not allowed to answer such questions to prevent unfair evaluation.

Wh*ere to start?* Check the exercises, we are making use of the information needed there, and we used them and discussed them in class.

I forgot XYZ in the delivery, can I change it? You will have time to change it till the deadline, and past the deadline, the assignment will be as delivered. Use the editing link from the delivery, if you lose it I can’t generate it for you.

I forgot to fill in the name of my partner, can I add it? You will have time to change it till the deadline. Past the deadline, the assignment will be considered as delivered, with or without edits.

Last note: Remember that some instructions are constrained to be in the position they are, which means there are no alternatives.