[**Individual Assignment**](https://bb-gbc.blackboard.com/webapps/assignment/uploadAssignment?content_id=_6317760_1&course_id=_208763_1&group_id=&mode=view)

For this assignment you must create a smart contract or smart contract library using the advanced features you have learned in this course.

Requirements

At your option, choose one of the following requirement streams:

1. Usage of assembly for non-trivial functionality. This can either be a full smart contract, or a tested library. It must also include benchmarking or gas profiling information.
2. Usage of off-chain computation such as signatures or merkle trees. This should be a tested, full smart contract along with an off-chain component.
3. Usage of Decentralized Finance (DeFi) concepts such as automated market makers or lending protocols. This should have a tested smart contract and description of performs a financial function.

Your submission must fulfill all the following requirements:

* Includes tests, using truffle or similar framework.
* Minimum 100% code coverage
* Contain a README file that explains the high-level design, implementation details, gas cost optimizations, security considerations, and anything else you think is relevant. Include as much detail as possible.
* Whenever you are ready, zip the project and make a submission. Copy the information contained in your README and paste it to submission text as well.
* Remember not to include **node\_modules** folder in your zipped project.

Grading

The grading break-down is as follows, out of a total of 100 points:

* 50 points : Fulfills either of the two requirement streams
  + Use of assembly, off-chain computation, or decentralized finance makes sense
  + Contract and all supporting material is included in the zipped folder, compiles and deploys correctly
  + More difficult or ambitious projects will earn higher points
* 12.5 points: Testing
  + Contract is well unit tested using truffle or similar
* 12.5 points: Documentation
  + Thorough documentation that describes your design
* 12.5 points: Security
  + Your smart contract should not have any security bugs
  + Documentation that describes the different attacks you have considered, and your contract prevents them
* 12.5 points: Efficient
  + Your smart contract should not consume any more gas than needed for its functionality
  + Documentation that describes what optimizations you have done, and how you have checked that your contract/library doesn't use excess gas

Ideas (assembly stream)

* Assembly to implement string operations which are not possibly with normal solidity
* Contracts that interact with other contracts (may require assembly)
  + [Upgradeable smart contracts](https://medium.com/coinmonks/summary-of-ethereum-upgradeable-smart-contract-r-d-part-2-2020-db141af915a0)
* Custom storage layout (requires assembly to bypass solidity layouts)
  + [Diamond layout](https://medium.com/1milliondevs/new-storage-layout-for-proxy-contracts-and-diamonds-98d01d0eadb), [Also this](https://hiddentao.com/archives/2020/05/28/upgradeable-smart-contracts-using-diamond-standard)
* Find a common smart contract operation, and make it more efficient using assembly

Ideas (off-chain stream)

* [Efficient token Air drop using merkle trees](https://blog.ricmoo.com/merkle-air-drops-e6406945584d)
* Voting system using off-chain voting roll call
* Ability to validate chunks of a large file on-chain without uploading the file to the chain, just a merkle root
* [Off-chain orderbook](https://www.codementor.io/@yosriady/signing-and-verifying-ethereum-signatures-vhe8ro3h6) using ecrecover() to validate orders
* Multi-signature wallet that requires multiple participants to sign a message

Ideas (DeFi)

* Custom Uniswap that uses experimental new demand curve
* NFT that can be borrowed, but charges interest the longer you hold it
* Way to move funds between different uniswap pools depending on where it is most profitable
* Experiment with different interest rate models