Homework Assignment 2

- Due: 11:59pm on Monday, November 3, 2025 (Pacific Time)
    - Late Submission Due: 11:59pm on Monday, November 10, 2025 (Pacific Time)
- Submit via Gradescope (Course Entry Code: VW3K2R)
- Starter Code: hw2-starter.zip
    - Foundry is required in order to build, run and test the code if you built on top of the starter code.
- Contact *Yanju Chen* on slack if you have any questions.

Overview and Getting Started

Gas optimization is a common and important topic in Solidity development. Because every operation on the Ethereum Virtual Machine (EVM) consumes gas, efficient smart contract design directly affects both cost and performance. Developers are encouraged to write cleaner, more optimized code that reduces unnecessary computation, storage, and external calls.

File and Folder Structures

In this homework, you will explore typical gas optimization techniques and learn how small code changes can lead to significant savings in deployment and execution costs. There are 4 case scenarios that you can find in the `src/` folder:

```
src/
  Bank.sol                  <-- abstract and original contract
  BankOptimized.sol         <-- put your optimized code here
  LayoutPlay.sol            <-- abstract and original contract
  LayoutPlayOptimized.sol   <-- put your optimized code here
  TicketSale.sol            <-- abstract and original contract
  TicketSaleOptimized.sol   <-- put your optimized code here
  UserPrefs.sol             <-- abstract and original contract
  UserPrefsOptimized.sol    <-- put your optimized code here
```

Each case contains an abstract contract that defines the API endpoints, and an original contract that implements the functionality. Your task is to explore and apply different gas optimization techniques so that the gas consumption of those functionalities can be reduced. The optimized version should be put in the `??Optimized.sol` contracts, as shown in the above.

Metrics

The starter pack comes with some Foundry test cases that you can use for debugging. They are located in `test/`. There are two evaluation metrics in this homework, namely: **functional correctness and gas consumption**.

To test the correctness your implementation (i.e., gas optimization should not change the semantics of the original implementation), simply invoke Foundry from the root path of the starter pack and issue the following command:

```
forge test
```

You will then see a series of automated testing running for evaluating functional correctness.

To test and compare the gas consumption, you'll need to utilize the gas reporting feature in Foundry (that's also how the autograder measure the gas cost when evaluating your submissions). There are special test cases that are designed to evaluate gas consumption in each scenario, namely: `testGas_OpenAccount_Log`, `test_abt`, `testGas_Buy_And_Withdraw_Log` and `test_Gas_Naive_vs_Optimized`. For example, if you want to test the simple use the following command:

```
forge test --match-test testGas_OpenAccount_Log --gas-report
```

You can see something like the following:

```
[⠒] Compiling...
No files changed, compilation skipped

Ran 1 test for test/Bank.t.sol:BankTest
[PASS] testGas_OpenAccount_Log() (gas: 147569)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 6.68ms (1.20ms CPU time)

┌-----------------------------------+-----------------+-------+--------+-------+---------
┐
| src/Bank.sol:BankOriginal Contract |                 |       |        |       |
|
+=============================================================================================
+
| Deployment Cost                    | Deployment Size |       |        |       |
|
|-----------------------------------+-----------------+-------+--------+-------+---------
|
| 718613                             | 3124            |       |        |       |
|
|-----------------------------------+-----------------+-------+--------+-------+---------
|
|                                    |                 |       |        |       |
|
|-----------------------------------+-----------------+-------+--------+-------+---------
|
| Function Name                      | Min             | Avg   | Median | Max   | # Calls
|
|-----------------------------------+-----------------+-------+--------+-------+---------
|
| openAccount                        | 66613           | 66613 | 66613  | 66613 | 1
|
```

```
└-----------------------------------+-----------------+-------+--------+-------+---------
┘


┌-------------------------------------------+-----------------+-------+--------+-------
+---------┐
| src/BankOptimized.sol:BankOptimized Contract |                 |       |        |
|         |
+=============================================================================================
==========+
| Deployment Cost                           | Deployment Size |       |        |
|         |
|-------------------------------------------+-----------------+-------+--------+-------
+---------|
| 718601                                    | 3124            |       |        |
|         |
|-------------------------------------------+-----------------+-------+--------+-------
+---------|
|                                           |                 |       |        |
|         |
|-------------------------------------------+-----------------+-------+--------+-------
+---------|
| Function Name                             | Min             | Avg   | Median | Max
| # Calls |
|-------------------------------------------+-----------------+-------+--------+-------
+---------|
| openAccount                               | 66613           | 66613 | 66613  | 66613
| 1       |
└-------------------------------------------+-----------------+-------+--------+-------
+---------┘


Ran 1 test suite in 12.04ms (6.68ms CPU time): 1 tests passed, 0 failed, 0 skipped (1
total tests)
```

You can then compare the gas consumption in the original contract `BankOriginal` and the optimized one `BankOptimized`. In the above example, as all the optimized versions provided in the starter pack are using the same code from the original version, so you can't actually see a difference in the gas consumption.

Target Functions to Optimize

For gas consumption, the following functions need to be optimized (i.e., their optimized versions should consume less gas than the original version):

- `openAccount(uint8)` from `BankOptimized.sol`
- `bumpABT(uint256,uint128,uint64)` from `LayoutPlayOptimized.sol`
- `buy(uint256)` from `TicketSaleOptimized.sol`
- `withdrawProceeds()` from `TicketSaleOptimized.sol`
- `toggle(uint256[])` from `UserPrefsOptimized.sol`

Submission and Evaluation

You will be sumitting the optimized version of the 4 cases via Gradescope, i.e.: `BankOptimized.sol`, `LayoutPlayOptimized.sol`, `TicketSaleOptimized.sol`, `UserPrefsOptimized.sol`. You can build upon the provided template in the starter pack, or build entirely on independently on your own --- either way is great, but *make sure your submitted smart contract has the same signature for each function* as defined in their corresponding abstract contracts. The autograder relies on the function signatures to locate and test different usage defined in the specification.

The maximum points you can get is 76 pts (14 test cases), which break down to:

- 10 pts (1 test case): submission of all required files.
- 16 pts (3 correctness test cases + 1 gas test case): test suite of the `Bank` contract.
- 10 pts (1 gas test case): test suite of the `LayoutPlay` contract.
- 30 pts (5 correctness test cases + 2 gas test cases): test suite of the `TicketSale` contract.
- 10 pts (1 gas test case): test suite of the `UserPrefs.sol` contract.

You get different amount of pts for each test case passed based on each one's importance.

Note that:

- The extra 10 pts won't be awarded if you only submit an empty solution; so please do try your best to understand the problem and write down the solution.
- There's a grace period of 1 hour after the regular deadline. Note that the grace period is intended for you to wrap up and save your unfinished work for submission at the last minute, and if you could not catch that, your subsequent submissions will be marked as late submissions.
- The points you get will be converted to percentage when computing score of this homework towards your final grade. For example, if you get 76/76 pts, you get 100% of the homework score, and if you get 38/76 pts, you'll get 50% of the homework score, etc.
- Points earned from late submission will be discounted by 50%. Late submission is due 1 week after the regular deadline. Subsequent submissions after the late submission deadline will receive 0 point. There's no grace period of late submission deadline.

Hints and Useful Resources

- You may find this Solidity language examples useful: https://solidity-by-example.org/
- Foundry itself is a tool that provides powerful testing utilities for Solidity. Check out its documentation here: https://book.getfoundry.sh/

Academic Integrity

Please refer to UCSB's adacemic integrity guidance (here) if you have any questions.