Cluster 1 HM

1)What are the concepts?

2)What is the organization?

3)What is the contract doing? What is the mechanism?

4)How could it be better? More efficient? Safer?

**Acct Check Native**

**1)**

-Conditionals, functions, system\_program, match function (let, return)

Conditionals: Acct Check Native extensively used if else clauses to check ids, account lengths, account creation through lamports, account changes on owners and keys. There were msg! and return is used to check conditions and send messages depending on the outcome of the contract.

Functions: In computer programming, everything serves as a function but the main function is process\_instruction. On process\_instruction, system\_program has been used for accounts, pubkeys, account infos, instruction data.

System\_program: System program is the main part of solana contracts and it does not work without adding system program for each smart contract on Solana. It has been used a few times when imported through solana library and being used on process\_instruction function.

Match functions: According to the contract, let and match functions were used extensively. These functions include accounts\_iter, \_payer, account\_to\_create, account\_to\_change and system\_program. After the use of let, conditionals were created to check the match functions, but return is used to check conditionals.

**2)**

The code organization broadly consists of the use of solana\_program, entrypoint, process\_instruction, msg!, conditionals, system\_program and match functions.

Solana program is declared at the first lines and it has imported account\_info, entrypoint, msg, program\_error, pubkey and system\_program. After the library and the relevant functions are imported, entrypoint is declared and a function called process\_instruction is formed.

**3)**

According to the code, this program is used to check accounts on native Solana program. It requires the system program, account info, public keys, program ID, account creation, initialization, modification and adding public keys.

The contract uses error handling on try/catch methods. Instead of using try/catch method, they return Errors named as ProgramError.

**4)**

For contract safety, it is recommended for native Solana developers/engineers to use borsh and other security mechanisms created by Solana Program Library (SPL). In addition to the use of borsh in native Solana, tests can be used with borsh and additional security tests could be written before deploying the Solana program. Instead, the test functions were comprehensively tested and JSON keypairs are produced as a result. Borsh could be added by writing use borsh::{BorshDeserialize, BorshSerialize};.

For the efficiency, I cannot say anything about it as I found all the functions have been used and all the functions were used.

**CPI NAT HAND**

**1)**

-System\_program, entrypoint, function, match, vector, clone and invoke has been used.

System\_program: System program is the main part of solana contracts and it does not work without adding system program for each smart contract on Solana. It has been used a few times when imported through solana library and being used on pull\_lever function.

Entrypoint: In Native Solana program, entrypoint has to be declared before the function with the pull\_lever parameter.

Match function: let function is used extensively while no conditional was used with mutable accounts. According to the program, accounts\_iter was used with mutable accounts as users can create accounts with the help of Solana system program. The functions used in this section are accounts\_iter, power, level\_program, set\_power\_status\_instruction, and ix.

Vector: At the end of the program with botsh, AccountMeta was used with a vector, with a Boolean variable, this vector check with Borsh that the instruction should give a false about key clone.

Clone: A vector is used to check whether power status works with the use of AccountMeta under the ix match function.

Invoke: Unlike result and Error functions, invoke is used at the end of the program as no conditionals were used.

Functions: Process\_instruction has been used as the only function.

**2)**

The code organization consists the use of lever, borsh and solana program being imported. It has been seen that Borsh has been deserialized, account infos, entrypoint, instructions, pubkey and program has been mentioned on the imported solana program library. The power status had been set and entrypoint is declared with pull\_lever function. After the main function is created, program\_id, accounts, instruction\_data has been created with program result and five match functions were used while the last one has a vector.

**3)**

According to the code, this program pulls the lever with instructions to check power. In other words, the contracts use power and pulls the lever automatically.

**4)**

In order to check whether match functions work, it is always good to add conditionals to check them and give message according to results. However, there is a concern that conditional functions can easily make the program size much better and it should be used sparingly so that conditionals are checked and appropriate output can be delivered and when necessary Error message can be given to the user, so an error function can be created and the Ok(()) output could be used to finalize the contract. Finally, try/catch methods can be actively used for error handling.

**CPI NAT LEVER**

**1)**

The concepts used in this program consists of functions, entrypoint, import libraries, macros, functions match functions, serialization, structs, strings, Boolean variables.

Library imports: Solana program library (SPL) and borsh has been imported for this Solana program (contract).

Macros: According to the program, #[cfg(not(feature=”no-entrypoint”))], #[derive(BorshDeserialize, BorshSerialize, Debug))] has been used. These macros are convenient for most developers as they can activate important parts of code, make debugging easier and it makes the use of imported libraries easier. The first macro shows that the feature has entrypoint and Borsh can Serialize and deserialize the data and debug when necessary.

Functions: It has the functions named initialize, switch\_power, and process\_instruction. But on entrypoint, only process\_instruction was written.

Match: Multiple match and let functions were extensively used inside the functions as conditionals.

Serialization: Borsh has been used to serialization and deserialization. According to macros, structs have been used to check Boolean values when borsh being used.

Structs: Two public structs have been used for power status while the first one sets the power status. These are declared as public to be reached out for external users.

Strings: Strings are used inside functions as names and public keys, this is the main data structure.

Boolean variables: PowerStatus struct, match power\_status and other match functions used Boolean variables to check whether power was on or off.

Functions: Process\_instruction, initialize, switch\_power has been used. Multiple functions were defined with matches to ensure all functions work on a simultaneous order.

**2)**

The code mainly consists of imported libraries, three different functions and structs in the end. The functions consists of standard solana instructions in the SPL and the match functions with serialization functions. But structs have been used as the final components.

**3)**

The contract sets power status, instructs the process, initialization, switching power. It is a power switch contract.

**4)**

For efficiency, conditionals can be used to check whether the values can be returned and an effective error handling can be established.