Lab 15-7-2023 (Part - 2)

Decision Making Statements

In Solidity, decision-making statements allow the program to execute different blocks of code based on certain conditions.

The two commonly used decision-making statements in Solidity are as following,

1.1 if/else statement:

- The if/else statement in Solidity enables the program to execute a specific block of code
- If a condition is true, and a different block of code if the condition is false.

Syntax:

```
if (condition) {
   // Code to be executed if the condition is true
} else {
   // Code to be executed if the condition is false
}
```

The condition within the if statement is evaluated, and if it results in true, the code within the if block is executed.

If the condition is false, the code within the else block (if present) is executed.

Example:

```
function checkNumber(uint256 _number) public pure returns (bool) {
    if (_number == 0) {
        return true;  // Execute this code if _number is 0
    } else {
        return false;  // Execute this code if _number is not 0
    }                                                          }
```

TASK

Design a Solidity contract named **AgeVerifier** that verifies if a person is eligible to vote. Implement a function named **checkEligibility** that takes an integer parameter age and returns a boolean value indicating whether the person is eligible to vote (age >= 18) or not.

1.2 if/else if/else statement:

- The if/else if/else statement allows the program to make decisions among several options.
- It checks multiple conditions in a sequence and executes the block of code associated with the first true condition.
- If none of the conditions are true, the code within the else block (if present) is executed.

Syntax:

```
if (condition1) {
   // Code to be executed if condition1 is true
} else if (condition2) {
   // Code to be executed if condition2 is true
} else {
   // Code to be executed if none of the conditions are true
}
```

The conditions are evaluated sequentially, and as soon as a condition is true, the corresponding block of code is executed.

If none of the conditions are true, the code within the else block is executed.

Example

```
function checkNumber(uint256 _number) public pure returns (string memory)
{
    if (_number > 10) {
        return "Number is greater than 10";
    } else if (_number < 10) {
        return "Number is less than 10";
    } else {
        return "Number is equal to 10";
    }
}
```

In the above example, the function checks the value of _number and returns a different string message based on its value.

- ✔ Decision-making statements like if/else and if/else if/else are essential in Solidity as they allow smart contracts to perform different actions based on specific conditions.
- ✔ They provide flexibility and control flow within the contract, enabling developers to implement various logical operations and decision-making processes.

TASK

Design a Solidity contract named **IntegerChecker** that takes integers (both +ve and -ve numbers) as input and recommends the data type (uint, int) as output.

Hint:

- Consider input integer between -255 to 255
- If input is positive, consider it **unit** and if negative, consider it **int.**

3.4 Ternary Operator (? :)

- The ternary operator is a concise and powerful operator in Solidity that allows for conditional execution of code based on a specified condition.
- It is the only operator in Solidity that takes three operands: a condition, an expression to execute if the condition is true, and an expression to execute if the condition is false.

Syntax:

```
condition ? expression_if_true : expression_if_false;
```

The ternary operator is commonly used as a compact alternative to an if-else statement when a simple decision needs to be made.
It provides a way to determine which expression should be evaluated based on the outcome of a condition.

TASK

Implement the first contract named **AgeVerifier,** now using Ternary Operator.

TASK

Create a Solidity function that takes an integer parameter representing a student's score in an exam, and returns **(bool, string).**

- If the score is greater than or equal to 60, return "True."
    - ➔ If score is between 80-100, return “Achieved Gade : A”
    - ➔ If score is between 60-79, return “Achieved Gade : B”
- If the score is less than 60, return "False."
    - ➔ If score is between 40-59, return “Achieved Gade : C”
    - ➔ For remaining 0-39, return “Better Luck Next Time”.