// SPDX-License-Identifier: MIT

// This is an Open-source License

// The version of Solidity being used; ensures that the code works with version 0.8.0 or higher

// Used for preventing compatibility issues.

pragma solidity ^0.8.0;

// Defining a contract named StudentRecords

contract StudentRecords

{

    // Struct to store student information

    struct Student

    {

        string name;

        uint age;

        string course;

    }

    // Array to hold the list of students

    Student[] public students;    // Syntax: datatype[] access specifier nameOfArray

    //  It is public, meaning anyone can read (but not modify) the list of students.

    // Mapping to track student index by name for easy removal

    mapping(string => uint) private studentIndex;

    // String student-name from students array is mapped to unsigned integers (index) for faster retrieval

    // It is private so that no one outside the contract can access this.

    // Function to add a new student

    function addStudent(string memory \_name, uint \_age, string memory \_course) public

    // The memory keyword indicates that the function parameters are stored temporarily in memory

    // (instead of on the blockchain, which uses the more expensive storage).

    {

        students.push(Student(\_name, \_age, \_course)); // Push the value (type Student) into the students array

        studentIndex[\_name] = students.length - 1; // Store index for easy removal

    }

    // Function to remove a student by name

    function removeStudent(string memory \_name) public

    {

        require(students.length > 0, "No students to remove."); // Empty array

        uint index = studentIndex[\_name];

        require(index < students.length, "Student does not exist.");

        // Move the last student into the place of the removed student to maintain array integrity

        students[index] = students[students.length - 1];

        // Update index in the mapping

        studentIndex[students[index].name] = index;

        students.pop(); // Remove the last element

    }

    // Fallback function to receive Ether and log it

    fallback() external payable

    // fallback func is called

    // (i) Func doesn't exist

    // (ii) Ether sent WITH data, without specific func call

    {

        // Simple logging mechanism for the fallback function

        emit LogFallback(msg.sender, msg.value);

        // emit is used to trigger an event; in this case the event id LogFallback

        // msg.sender: Address of the account the sent the ether or request for func call

        // msg.value: Amt of ether sent if any

    }

    // Receive function to receive Ether directly

    receive() external payable

    // It is only triggered when Ether is sent WITHOUT data.

    {

        emit LogReceived(msg.sender, msg.value);

    }

        /\*

        Use only receive() if you just want to handle simple Ether transfers (no data, no unknown function calls).

        Use fallback() if you want to handle situations where Ether might be sent with additional data,

        or if you need to catch invalid function calls.

        \*/

    // Function to check the balance of the contract

    function getBalance() public view returns (uint)

    // view: this function does not modify any state; It only reads from the blockchain ("non-transactional" operation)

    {

        return address(this).balance; //measured in Wei (the smallest unit of Ether).

    }

    // Payable function to allow users to pay for adding a student record

    function payForStudentRecord(string memory \_name, uint \_age, string memory \_course) public payable

    {

        require(msg.value > 0, "You need to send some Ether to add a student record.");

        students.push(Student(\_name, \_age, \_course));

        studentIndex[\_name] = students.length - 1;

    }

    // Event to log fallback transactions

    event LogFallback(address indexed sender, uint value);

    // Event to log received Ether

    event LogReceived(address indexed sender, uint value);

}