Aim: Write a program in solidity to create Student data. Use the following constructs:

- Structures
- Arrays
- Fallback

Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values.

Objective:

- Get familiar with solidity language with the help of few simple programs

Requirement:

- Remix IDE

Solidity

- Solidity is a contract-oriented, high-level programming language for implementing smart contracts. Solidity is highly influenced by C++, Python and JavaScript and has been designed to target the Ethereum Virtual Machine (EVM).
- We can use Solidity to create contracts for uses such as voting, crowdfunding, blind auctions, and multi-signature wallets.

```
pragma solidity ^0.8.0;

contract StudentData {

   // Structure to hold student details
   struct Student {
      uint rollNo;
      string name;
      uint marks;
   }

   // Dynamic array to store students
   Student[] public students;

   // Add a new student to the array
   function addStudent(uint _rollNo, string memory _name, uint _marks) public {
        Student memory newStudent = Student({
            rollNo: _rollNo,
            name: _name,
            marks: _marks
        });

        students.push(newStudent);
    }

    // Get details of a student by index
    function getStudent(uint index) public view returns (uint, string memory, uint) {
        require(index < students.length, "Index out of bounds");
        Student memory s = students[index];
        return (s.rollNo, s.name, s.marks);
    }

    // Get total number of students
    function getTotalStudents() public view returns (uint) {
        return students.length;
    }

    // Fallback function - gets called when unknown function or ether is sent
    fallback() external payable {
        // You can log something or keep it empty
    }

    // Receive function to accept Ether (optional)
    receive() external payable {}
}
```

Explanation:

```
pragma solidity ^0.8.0;
```

- **pragma**: Specifies the version of the Solidity compiler.
- **^0.8.0**: The contract works with version 0.8.0 or above (excluding breaking changes in future versions).

```
contract StudentData {
```

- **contract**: Defines a smart contract named `StudentData`.
- This contract manages data of multiple students on the blockchain.

```
struct Student { uint rollNo; string name; uint marks; }
```

- **struct**: Used to define a custom data type named `Student`.
- **uint rollNo**: Stores the student's roll number (unsigned integer).
- **string name**: Stores the name of the student.
- **uint marks**: Stores the marks of the student.

```
Student[] public students;
```

- **Student[]**: A dynamic array to store multiple `Student` records.
- **public**: Anyone can view the `students` array.
- The array grows automatically as new students are added.

```
function addStudent(uint _rollNo, string memory _name, uint
_marks) public {
```

- **function addStudent**: Adds a new student to the list.
- **uint _rollNo**: Input roll number.
- **string memory _name**: Input name (stored in memory).
- **uint _marks**: Input marks.
- **public**: The function can be called by anyone.

```
Student memory newStudent = Student({...});
```

- Creates a temporary `Student` structure in memory with given details.

```
students.push(newStudent);
```

- Adds the newly created student to the `students` array.

```
function getStudent(uint index) public view returns (...)
```

- **getStudent**: Used to fetch the details of a student at a specific index.
- **uint index**: The index position of the student in the array.
- **public view**: Can be called by anyone, and it doesn't change the blockchain state.
- **returns**: It gives roll number, name, and marks.

```
require(index < students.length, "Index out of bounds");
```

- **require**: Checks if the index is valid.
- If not, the function fails with the message **"Index out of bounds"**.

```
return (s.rollNo, s.name, s.marks);
```

- Returns the selected student's roll number, name, and marks.

```
function getTotalStudents() public view returns (uint)
```

- Returns the total number of students in the array.

```
fallback() external payable { }
```

- A special unnamed function.
- **fallback**: Called when a wrong function is called or when ETH is sent without any data.
- **external**: Can be called from outside the contract.
- **payable**: It can receive Ether (though it doesn't use it here).

```
receive() external payable { }
```

- **receive**: Special function that accepts plain ETH transfers (with no data).
- **external payable**: Allows contract to receive Ether directly from users.

Conclusion:
We have learned how to write, compile, deploy and run a smart contract in solidity language using remix IDE.