BT4

🎯 PRACTICAL OVERVIEW

This Solidity smart contract creates and stores Student data using:

- Structures (struct)
- Arrays
- Fallback & Receive functions (to handle incoming Ether)

The contract allows:

- Adding a student
- Retrieving student details
- Counting total students
- Accepting Ether via fallback/receive and recording it in events

This contract is deployed on an Ethereum test network (e.g., Remix VM / Sepolia / Holesky / Ganache) and gas cost for transactions can be observed.

📝 PROBLEM STATEMENT

Write and deploy a smart contract that maintains student information using Structure and Array, and includes fallback and receive functions to handle Ether transactions.
Observe transaction fee and gas value while interacting with the contract.

Subject: Blockchain Technology (BT)
Topic: Smart Contracts with Structs, Array, and Fallback/Receive Ether Handling

---

🧠 CONCEPTS USED

Concept	Meaning
struct	User-defined data type used to group variables.
mapping	Key-value store for accessing student details.

Concept	Meaning
array	Used to track list of student IDs.
receive()	Special function triggered when contract receives Ether with empty calldata.
fallback()	Triggered when calldata is invalid or function does not exist. Also receives Ether.
event	Used to log transaction data on blockchain for traceability.

🔍 CODE EXPLANATION (Line-by-line)

```
// SPDX-License-Identifier: Bhide License
```

- SPDX license identifier (just a comment for licensing).

```
pragma solidity ^0.8.0;
```

- Use Solidity compiler version 0.8.0 or above (safe arithmetic included).

```
contract StudentRegistry {
```

- Start of smart contract named StudentRegistry.

```
    struct Student {
        int id;
        string name;
        string department;
    }
```

- struct Student groups student data into one type (integer ID, string name, string department).

```
    mapping(int => Student) private students;
```

- mapping stores students by their unique ID.
- private means only functions inside contract can directly access.

```
    int[] private studentIds;
```

- Dynamic array to store list of student IDs for counting the number of students.

```
event ReceivedEther(address indexed sender, uint256 value);
```

- Event to log whenever contract receives Ether.
- indexed makes sender searchable in logs.

```
receive() external payable {
    emit ReceivedEther(msg.sender, msg.value);
}
```

- receive() is automatically called when contract receives Ether without data.
- payable means it can receive Ether.
- Emits event with sender and value.

```
fallback() external payable {
    emit ReceivedEther(msg.sender, msg.value);
}
```

- fallback() runs when:
    - Function name does not exist
    - Or calldata is invalid
- Also marked payable to accept Ether.

```
function addStudent(int id, string memory name, string memory department) public {
    require(bytes(students[id].name).length == 0, "Student ID already exists");
    students[id] = Student(id, name, department);
    studentIds.push(id);
}
```

- Adds a student.

- require() ensures student ID is unique (name string length zero means not yet stored).
- Stores struct in mapping.
- Pushes ID into ID array.

function getStudent(int id) public view returns (int, string memory, string memory) {

require(bytes(students[id].name).length != 0, "Student not found");

Student memory s = students[id];

return (s.id, s.name, s.department);

}

- Returns student details.
- Uses view because it does not modify the state.
- Uses local memory copy for efficiency.

function getStudentCount() public view returns (uint256) {

return studentIds.length;

}

- Returns the number of students stored.
- Array .length gives count.

}

- End of contract.

📥 INPUT (Example using Remix)

Action	Function Call	Example Values
Add Student	addStudent(1, "Onkar", "Computer")	Adds new student
Get Student	getStudent(1)	Returns stored details

Action	Function Call	Example Values
Get Count	getStudentCount()	Returns total number of students
Send Ether	Enter value → click transact	e.g., 0.01 ether

📤 OUTPUT (What You See)

Function	Output
getStudent(1)	(1, "Onkar", "Computer")
getStudentCount()	Number of stored students
Sending Ether	Logged event: ReceivedEther(sender, amount)

⛽ GAS & TRANSACTION FEE OBSERVATION

- Adding a student modifies blockchain state → Costs Gas
- Retrieving student is read-only → Costs 0 Gas (if called view off-chain)
- Sending Ether via fallback generates event → Gas used
- Gas Used × Gas Price = Transaction Fee in ETH

You will see:

- Gas Used in Remix console
- Transaction Cost in test network explorer (if using MetaMask)

💡 EXAM TIPS & VIVA QUESTIONS

Question	Short Answer
What is a struct?	User-defined datatype used to group related variables.
Why use mapping?	Fast key-value storage; O(1) lookup.

Question	Short Answer
Why use array along with mapping?	Mapping does not store keys; array helps count/iterate student IDs.
Difference between fallback and receive?	receive() is called when ether sent without data, fallback runs otherwise.
Why emit events?	Helps track logs and verify transactions on blockchain explorer.
Does getStudent() use gas?	No, because it is a view function (if called off-chain).

🏋️ PRACTICE MODIFICATIONS (For viva confidence)

1. Add updateStudent() function
2. Add deleteStudent() function
3. Store marks or CGPA in struct & update
4. Return full student list (explain gas considerations)

✅ Code with Each Line Explained

// SPDX-License-Identifier: Bhide License

- This is a license identifier comment.
- It tells tools what license the code uses.
- It does **not** affect contract execution.

pragma solidity ^0.8.0;

- This tells the compiler which Solidity version to use.
- ^0.8.0 means use **0.8.0 or higher**, but **below 0.9.0**.
- Version 0.8+ has built-in protection for integer overflow.

contract StudentRegistry {

- Declares a new smart contract named **StudentRegistry**.
- A contract is similar to a class in OOP.

```
struct Student {
    int id;
    string name;
    string department;
}
```

- struct is a **user-defined data type**.
- It groups related data together.
- Each student has:
    - id (integer),
    - name (string),
    - department (string).

```
mapping(int => Student) private students;
```

- mapping is like a dictionary / key-value store.

- Key = student id, Value = Student struct.
- private means only functions inside this contract can directly access it.

int[] private studentIds;

- studentIds is a **dynamic array**.
- It stores all student IDs in order to keep count of total students.
- private means only inside contract usage.

event ReceivedEther(address indexed sender, uint256 value);

- event is used to log information to the blockchain.
- Here, it logs when Ether is sent to the contract.
- indexed allows filtering event logs by sender address.

```
receive() external payable {
   emit ReceivedEther(msg.sender, msg.value);
}
```

- receive() is a **special function** automatically called when:
    - Someone sends **Ether** to contract,
    - With **no function call data**.
- external → can only be called outside the contract.
- payable → allows the contract to **accept Ether**.
- emit fires the event and records who sent Ether and how much.

```
fallback() external payable {
   emit ReceivedEther(msg.sender, msg.value);
}
```

- fallback() is called when:
    - A function that does **not exist** is called,
    - Or data is sent with the transaction.
- Also payable, so it accepts Ether too.
- Logs the action same as receive().

```
function addStudent(int id, string memory name, string memory department) public
{
```

- Function to **add a new student**.
- Parameters:
    - id: integer student ID
    - name: stored in memory during execution
    - department: stored in memory
- public → any user or contract can call it.

require(bytes(students[id].name).length == 0, "Student ID already exists");

- require enforces a condition.
- Checks if student does NOT already exist.
- We check by verifying name length = 0 (means empty, means student not added yet).
- If false → revert with error "Student ID already exists".

students[id] = Student(id, name, department);

- Stores the new student in the mapping.
- Creates a new Student struct and assigns it to key id.

studentIds.push(id);

- Saves ID in array to keep track of total students.
- push() adds a value at end of array.

}

- End of addStudent() function.

function getStudent(int id) public view returns (int, string memory, string memory)
{

- Function to **read** student details.
- view → means it does not change contract data.
- Returns student ID, name, department.

require(bytes(students[id].name).length != 0, "Student not found");

- Checks if student **exists**.
- If name is empty, student was never added → revert with error.

Student memory s = students[id];

- Creates a temporary copy s of student data in memory.

  return (s.id, s.name, s.department);

- Returns the values stored in struct.

}

- End of getStudent() function.

function getStudentCount() public view returns (uint256) {

  return studentIds.length;

}

- Returns total number of students added.
- .length returns size of array.
- view means read-only and costs **no gas** when called locally.

}

- End of contract.

⭐ What Outputs You Will See in Remix

Function	Output Example
addStudent(1, "Onkar", "Computer")	No direct output; student is stored
getStudent(1)	1, "Onkar", "Computer"
getStudentCount()	1
Sending Ether to Contract	Event shows: ReceivedEther(sender, amount)

🎤 Viva Short Answers

Question	Answer
What is struct?	A user-defined composite datatype to store multiple values.
Why mapping + array?	Mapping stores data; Array allows counting and listing IDs.
When is receive() used?	When contract gets Ether **without data**.

Question	Answer
When is fallback() used?	When function does not exist OR data is sent.
Does getStudent() use gas?	No, it's a **view** read-only call.

🎯 Understanding INPUT & OUTPUT

In a smart contract, **INPUT** means:

- The **values** you give to a function when calling it.

And **OUTPUT** means:

- The **values returned** or **changes in blockchain state**.

✅ INPUT and OUTPUT for Each Function

Function Name	INPUT (What you give)	OUTPUT (What you get)	What Happens Internally
addStudent(int id, string name, string department)	Example: 1, "Onkar", "Computer"	No direct output (no return value)	Stores student data in mapping and pushes id in array
getStudent(int id)	Example: 1	Returns: (1, "Onkar", "Computer")	Reads student data from mapping
getStudentCount()	No input	Returns: total number of students (e.g., 1, 2, etc.)	Reads length of array
receive() (special)	Send Ether with **no function call**	No return value — logs event	Stores no data; simply triggers ReceivedEther event
fallback() (special)	Call contract with **invalid function name** or send Ether with data	No return value — logs event	Useful for trapping unknown calls

---

🧪 Example in Remix

1) Add Students (INPUT)

Call:

addStudent(1, "Onkar", "Computer")

addStudent(2, "Anjali", "IT")

→ No visible return values.

2) Get Student (OUTPUT)

Call:

getStudent(1)

You get:

1

"Onkar"

"Computer"

3) Count Students (OUTPUT)

Call:

getStudentCount()

You get:

2

4) Sending Ether (INPUT)

- Enter Value: 0.01 ether
- Click **Transact** (no function selected)

OUTPUT → Event log:

ReceivedEther(yourAddress , 10000000000000000)

🧠 One-Line Exam Answer

Input = Values passed to functions (student ID, name, department, or Ether amount).
Output = Returned student data or count, and event logs when Ether is received.

Would you like me to now prepare:

✓ **Viva speech (30–45 seconds)**

✓ **Gas / Transaction Fee explanation for this contract**

✓ **Remix deployment steps with screenshots**

Just say **yes** 😊