Student data contract

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.16;

contract StudentContract
{
   struct Student
   {
      uint stud_id;
      string name;
      uint marks;
   }
   Student[5] s;
   uint cnt = 0;
   constructor()
   {
      for(uint i=0;i<5;i++)
      {
         s[i].stud_id = 0;
         s[i].name = "";
         s[i].marks = 0;
      }
   }
   function getData(uint id) public view returns(string memory,uint)
   {
      /* Calling a revert statement implies an exception is thrown,
      the unused gas is returned and the state reverts to its original state. */
      if(id > cnt)
         revert("Invalid STUDENT ID");
      else
      {
         for(uint i=0;i<5;i++)
```

```
        {
            if(s[i].stud_id == id)
                return (s[i].name,s[i].marks);
        }
    }
    return ("", 0); // Default return statement
}
function setData(string calldata nm,uint mk) public returns(string memory)
{
    /* Calling a revert statement implies an exception is thrown,
    the unused gas is returned and the state reverts to its original state. */
    if(cnt > 5)
        revert("ARRAY IS FULL");
    else
    {
            cnt += 1;
            s[cnt-1].stud_id = cnt;
            s[cnt-1].name = nm;
            s[cnt-1].marks = mk;
    }
    return "Student record added.";
}
function search(uint id) public view returns(string memory,uint)
{
    /* Calling a revert statement implies an exception is thrown,
    the unused gas is returned and the state reverts to its original state. */
    if(id > cnt)
        revert("Invalid STUDENT ID");
    else
    {
        for(uint i=0;i<5;i++)
        {
```

```
            if(s[i].stud_id == id)
            {
                return (s[i].name,s[i].marks);
            }
        }
        return ("",0);
      }

   }
    /* This function is called for all messages sent to this contract,
    except plain Ether transfers any call with non-empty calldata to this contract
    will execute the fallback function (even if Ether is sent along with the call)*/
   fallback() external payable
   {

   }

   receive() external payable {
   // handle received ether here
   }
}
```