This challenge tests the ability to read, understand and use the provided implementation of the ATM class which follows a state-machine design pattern, to achieve the following behavior.

The ATM has two possible states:

- On successful login, a user's state is *authorized*.
- Before a successful login and after a successful logout, the user's state is *unauthorized*. Initially, the user's state is *unauthorized*.

In the snippet, initially, *a* has not logged in so its state is *unauthorized.*

```
>>>a = ATM()
>>>print(a)
unauthorized
```

Implement transitions of the ATM. Specifically, define the *transition_table* variable to be a Dictionary where keys are State objects and the values are lists of transitions supported in that state. Each transition is a tuple of 3 elements: *<action_name>, checker, next_state* and indicates that this transition should be used if the *action_name* matches the current action.

The checker has the following signature.

```
checker(action_param: Optional, atm_password: str, atm_current_balance:int ) -> Tuple[bool, int,
Optional]
```

In the returned Tuple:

- The first element denotes whether this transition should be performed or not.
- The second element denotes the balance of the ATM after the transition.
- The third element denotes the value returned by the transition (relevant only when fetching the balance).

The *next_state* object denotes the state the ATM should transition to if the transition is performed.

Actions are given as string inputs. Each string contains an action, a space character, and its parameter if any. There are five possible actions.

This is the only action that must be handled when the state is *unauthorized.*

- *login <password>* - There is no return value.
  - If the *password* is correct, the request is successful and the user's new state is *authorized*.

    - If the *password* is incorrect, the request is unsuccessful and the user's state remains *unauthorized.*

These are the actions that must be handled when the state is *authorized.*

- *logout* - The request is successful and the user's new state is *unauthorized.* There is no return value.
- *deposit <amount>* - The request is successful and the *amount* is added to the account balance.
- *withdraw <amount>* -
    - If the *balance* in the account is at least as much as the requested *amount*, the request is successful. The *amount* is deducted from the balance.
    - Otherwise, the request is unsuccessful and the balance is unchanged.
- *balance* - return the balance

Remember that a part of this challenge is testing the ability to understand and use the provided code. In case of any doubts, refer to the provided code of the ATM class to see how it uses states and transitions.

```
class ATM:
    def __init__(self, init_state: State, init_balance: int, password: str, transition_table: Dict):
        self.state = init_state
        self._balance = init_balance
        self._password = password
        self._transition_table = transition_table

    def next(self, action: Action, param: Optional) -> Tuple[bool, Optional]:
        try:
            for transition_action, check, next_state in self._transition_table[self.state]:
                if action == transition_action:
                    passed, new_balance, res = check(param, self._password, self._balance)
                    if passed:
                        self._balance = new_balance
                        self.state = next_state
                        return True, res
        except KeyError:
            pass
        return False, None
```

Your implementation will be tested by a provided code stub on several input files. Each input file contains the password for the ATM and its initial balance. Then, it contains descriptions of the actions that will be performed on the ATM by the provided code. The result of each transition is printed to the standard output by the provided code.

Input Format Format for Custom Testing
Input from stdin will be processed as follows and passed to the function.

In the first line, there is a single string, *password,* the correct password.
In the second line, there is a single integer, *init_balance,* the initial balance of the ATM.
In the third line, there is a single integer, *q*, the number of actions the ATM will be tested on.
Next, *q* lines follow. Each of them describes a single action request.

Sample Case 0
Sample Input

```
hacker
10
8
withdraw 5
login foo
login hacker
withdraw 15
deposit 20
withdraw 15
balance
logout
```

Sample Output

```
Success=False unauthorized
Success=False unauthorized
Success=True authorized
Success=False authorized
Success=True authorized
Success=True authorized
Success=True authorized 15
Success=True unauthorized
```

Explanation
The ATM's password is "hacker" and its initial balance is 10. There are 8 actions to be performed. The first action tries to withdraw 5 from the ATM but since it always starts in the "unauthorized" state, this action is unsuccessful. The second action tries to login with password "foo". This does not match the ATM's password so the action is also

unsuccessful. The third action tries to login with the correct password "hacker". This action is successful so the ATM moves to the "authorized" state. The next action tries to withdraw 15 from the ATM but this amount is greater than the current balance of the ATM so the action is unsuccessful. The fourth action deposits 20 to the ATM so its current balance is now 10+20=30. The next action, again, tries to withdraw 15 and it successfully changes the balance of the ATM to 30-15=15. The next action gets the current balance of the ATM, i.e. 15. The last action logs out so the ATM moves to the "unauthorized" state.

Sample Case 1

Sample Input

```
somepass
100
5
logout
login somepass
balance
login somepass
login wrongpass
```

Sample Output

```
Success=False unauthorized
Success=True authorized
Success=True authorized 100
Success=False authorized
Success=False authorized
```

Explanation

The ATM's password is "somepass" and its initial balance is 100. There are 5 actions to be performed. The first action tries to logout but since the ATM always starts in the "unauthorized" state, this action is unsuccessful. The second action tries to login with the correct password "somepass". This action is successful so the ATM moves to the "authorized" state. The next action gets the balance of the ATM so 100 is returned. The next two actions both try to login but since the ATM is already in the "authorized" state, both of them are unsuccessful.