LAB 8:

Create a knowledge base consisting of FOL statements and prove the query using forward reasoning.

```
class ForwardChaining:
    def __init__(self):
        self.facts = set()  # Store known facts
        self.rules = []     # Store rules as (premise, conclusion)

    def add_fact(self, fact):
        """Add a fact to the knowledge base."""
        self.facts.add(fact)

    def add_rule(self, premise, conclusion):
        """Add a rule to the knowledge base."""
        self.rules.append((premise, conclusion))

    def apply_rule(self, rule):
        """Apply a rule and derive new facts."""
        premise, conclusion = rule
        if premise <= self.facts:  # Premise is a subset of facts
            self.facts.add(conclusion)  # Add the conclusion to the
facts
            return True
        return False

    def forward_chain(self):
        """Perform forward chaining to derive new facts."""
        new_facts = True
        while new_facts:
            new_facts = False
            for rule in self.rules:
                if self.apply_rule(rule):
                    new_facts = True

    def prove_crime(self, person):
        """Check if a person is a criminal."""
        return ('Crime', person) in self.facts


# Initialize the forward chaining engine
fc = ForwardChaining()

# Facts
fc.add_fact(('American', 'Robert'))  # Robert is an American
fc.add_fact(('Sold', 'Robert', 'Missiles', 'A'))  # Robert sold
missiles to Country A
fc.add_fact(('Hostile', 'A'))  # Country A is hostile
```

```
# Rule: If an American sells weapons to a hostile country, they are a
criminal
fc.add_rule(
    {('American', 'X'), ('Sold', 'X', 'Missiles', 'Y'), ('Hostile',
'Y')},  # Premise
    ('Crime', 'X')  # Conclusion
)

# Perform forward chaining
fc.forward_chain()

# Check if Robert is a criminal
if fc.prove_crime('Robert'):
    print("Robert is a criminal.")
else:
    print("Robert is a criminal.")
```

output: