```
#Implement unification in First Order Logic
def is_variable(x):
    """Checks if x is a variable (assuming variables are single lowercase letters)."""
    return isinstance(x, str) and x.islower() and len(x) == 1

def occurs_check(var, term):
    """Checks if a variable occurs in a term (used to avoid circular unification)."""
    if var == term:
        return True
    if isinstance(term, tuple):  # If term is a function (tuple), check its arguments.
        return any(occurs_check(var, t) for t in term)
    return False

def unify(x, y, substitution=None):
    """Unifies two terms x and y, applying substitutions."""
    if substitution is None:
        substitution = {}

    # Case 1: If both terms are the same, no unification needed
    if x == y:
        return substitution

    # Case 2: If x is a variable, try to unify
    elif is_variable(x):
        if x in substitution:
            return unify(substitution[x], y, substitution)
        elif occurs_check(x, y):
            raise ValueError(f"Unification fails due to occurs check for {x} in {y}")
        else:
            substitution[x] = y
            return substitution

    # Case 3: If y is a variable, try to unify
    elif is_variable(y):
        return unify(y, x, substitution)

    # Case 4: If both terms are compound (functions), unify their components
    elif isinstance(x, tuple) and isinstance(y, tuple):
        if x[0] != y[0]:
            raise ValueError(f"Unification fails: {x[0]} != {y[0]}")
        # Recursively unify arguments
        for a, b in zip(x[1:], y[1:]):
            substitution = unify(a, b, substitution)
        return substitution

    # Case 5: Unification fails if x and y have no other cases
    else:
        raise ValueError(f"Unification fails: {x} cannot be unified with {y}")

def apply_substitution(term, substitution):
    """Applies the substitution to the term."""
    if isinstance(term, str):
        return substitution.get(term, term)
    elif isinstance(term, tuple):
        return (term[0], *[apply_substitution(t, substitution) for t in term[1:]])
    return term

def parse_term(term_str):
    """Parses a string representation of a term into a Python data structure."""
    term_str = term_str.strip()

    # Case 1: If it's a variable (single lowercase letter)
    if term_str.islower() and len(term_str) == 1:
        return term_str

    # Case 2: If it's a constant (any non-empty string, for example 'apple')
    if term_str.isalpha():
```

```
            return term_str

        # Case 3: If it's a function, e.g., 'f(x, y)'
        if term_str.startswith('f(') and term_str.endswith(')'):
            func_str = term_str[2:-1]  # Remove 'f(' and ')'
            parts = func_str.split(',')
            return ('f', *[parse_term(p.strip()) for p in parts])  # Function name, arguments

        # If none of these, raise an error
        raise ValueError(f"Invalid term format: {term_str}")
    print('SWAPNIL SAHIL(1BM22CS300):')

    def main():
        print("Enter two terms to unify (e.g., f(x, y), f(a, b)):")
        term1_str = input("Enter first term: ")
        term2_str = input("Enter second term: ")

        try:
            term1 = parse_term(term1_str)
            term2 = parse_term(term2_str)

            print(f"Unifying terms: {term1} and {term2}")
            substitution = unify(term1, term2)

            # Apply substitution to both terms to get the unified expression
            unified_term1 = apply_substitution(term1, substitution)
            unified_term2 = apply_substitution(term2, substitution)

            print("Unification successful!")
            print("Substitution:", substitution)
            print("Unified expression:")
            print(f"Term 1 after substitution: {unified_term1}")
            print(f"Term 2 after substitution: {unified_term2}")

        except ValueError as e:
            print("Unification failed:", e)

    # Run the program
    if __name__ == "__main__":
        main()
```

```
SWAPNIL SAHIL(1BM22CS300):
Enter two terms to unify (e.g., f(x, y), f(a, b)):
Enter first term: f(x,car)
Enter second term: f(bike,y)
Unifying terms: ('f', 'x', 'car') and ('f', 'bike', 'y')
Unification successful!
Substitution: {'x': 'bike', 'y': 'car'}
Unified expression:
Term 1 after substitution: ('f', 'bike', 'car')
Term 2 after substitution: ('f', 'bike', 'car')
```