LAB 09:

**Implementation of unification in FLO**

code:-

```
def is_variable(term):
    """
    Check if a term is a variable.
    Variables are typically single lowercase letters.
    """
    return isinstance(term, str) and term.islower()


def unify(expr1, expr2, subst={}):
    """
    Unify two expressions expr1 and expr2 under the given substitution subst.
    """
    if subst is None:
        return None  # Failure case
    if expr1 == expr2:
        return subst  # Expressions are identical
    if is_variable(expr1):
        return unify_variable(expr1, expr2, subst)
    if is_variable(expr2):
        return unify_variable(expr2, expr1, subst)
    if isinstance(expr1, tuple) and isinstance(expr2, tuple):
        if len(expr1) != len(expr2):
            return None  # Different arity
        # Recursively unify each component
        for arg1, arg2 in zip(expr1, expr2):
            subst = unify(arg1, arg2, subst)
            if subst is None:
                return None  # Failure
        return subst
```

```
    return None  # No unification possible

def unify_variable(var, term, subst):
    """
    Unify a variable with a term, updating the substitution.
    """
    if var in subst:
        return unify(subst[var], term, subst)  # Apply substitution to var
    if term in subst:
        return unify(var, subst[term], subst)  # Apply substitution to term
    if occurs_check(var, term, subst):
        return None  # Circular substitution detected
    # Add var -> term to the substitution
    subst = subst.copy()
    subst[var] = term
    return subst

def occurs_check(var, term, subst):
    """
    Check if var occurs in term (directly or indirectly) to prevent circular substitutions.
    """
    if var == term:
        return True
    if isinstance(term, tuple):
        return any(occurs_check(var, t, subst) for t in term)
    if term in subst:
        return occurs_check(var, subst[term], subst)
    return False

def parse_input(expr):
    """
```

```
    Parse user input into a structured format (nested tuples for functions and terms).
    Example: "f(X, g(y))" -> ('f', 'X', ('g', 'y'))
    """
    expr = expr.strip()
    if '(' not in expr:
        return expr  # Simple variable or constant
    func_name = expr[:expr.index('(')].strip()
    args = expr[expr.index('(') + 1:expr.rindex(')')].split(',')
    args = [parse_input(arg.strip()) for arg in args]
    return (func_name, *args)

def format_output(expr):
    """
    Convert the nested tuple representation back into a string for output.
    Example: ('f', 'X', ('g', 'y')) -> "f(X, g(y))"
    """
    if isinstance(expr, str):
        return expr
    return f"{expr[0]}({', '.join(format_output(arg) for arg in expr[1:])})"

# Main Program
if __name__ == "__main__":
    print("Enter the first term:")
    expr1 = parse_input(input().strip())
    print("Enter the second term:")
    expr2 = parse_input(input().strip())

    print("Unifying.....")
    result = unify(expr1, expr2)

    if result is None:
```

```
        print("Unification failed")
    else:
        print("Unification succeeded with substitution:")
        for var, term in result.items():
            print(f"{var} -> {format_output(term)}")
```

Output:-