**LAB SESSION 4: FIRST AND FOLLOW OF A CONTEXT FREE GRAMMAR**

**AIM**: To implement a program in python to compute the FIRST and FOLLOW of a given CFG.

**PROBLEM DEFINITION:** Develop a python program to compute the FIRST and FOLLOW of a given Context Free Grammar. Accept the Grammar from a file.

**THEORY:** The First and Follow sets are important in syntax analysis, mainly in parsing. All sets help in making predictive parsers and are integral to identifying how a given grammar can be parsed effectively.

The First Set for a non-terminal symbol represents all possible terminals that can appear at the beginning of any string derived from that non-terminal.

The follow set contains terminals that can appear immediately after a non-terminal in the derivation of the grammar.

All meaning of First and Follow sets is it allows to define which production rules are implemented in different parsing moment.

**What is the First Set?**

The First set of non-terminal in grammar tells us all the possible terminals that can appear at the beginning of any string derived from that non terminal. it gives us a way to know what symbol might come first if we start with a particular non-terminal. For example, if we have a rule like A -> aB | ε, then the First set of A includes the terminal a, as well as ε, which represents the possibility of deriving an empty string. Calculating the First set helps us predict which rules to apply during parsing by knowing which symbols can appear first in a derivation.

**What is a Follow Set?**

The Follow set of a non-terminal contains all terminals that can immediately follow it in any derivation of the grammar. This means that if a non-terminal appears in the middle of a sentence, the Follow set tells us what symbol might come right after it. For example, in a rule like S -> AB, the Follow set of A might include anything that can follow B, depending on other rules in the grammar. Follow sets are especially useful for parsers when making decisions about which rule to apply next and ensuring that we’re interpreting the structure of a sentence correctly, especially for LL(1) parsers where we only look one symbol ahead.

The follow set of the start symbol will always contain "$". Now the calculation of Follow falls under three broad cases:

If a Non-Terminal on the R.H.S. of any production is followed immediately by a Terminal then it can immediately be included in the Follow set of that Non-Terminal.

If a Non-Terminal on the R.H.S. of any production is followed immediately by a Non-Terminal, then the First Set of that new Non-Terminal gets included on the follow set of our original Non-Terminal. In case encountered an epsilon i.e. " # " then, move on to the next symbol in the production.

Note: "#" is never included in the Follow set of any Non-Terminal.

If reached the end of a production while calculating follow, then the Follow set of that non-terminal will include the Follow set of the Non-Terminal on the L.H.S. of that production. This can easily be implemented by recursion.

**PROGRAM:**

from collections import defaultdict

class FirstFollowCalculator:

    def \_\_init\_\_(self, grammar):

        self.grammar = grammar

        self.first\_sets = {}

        self.follow\_sets = defaultdict(set)

        self.first\_in\_progress = set()  # To handle left recursion

        self.follow\_in\_progress = set()  # To handle left recursion

        self.computation\_steps = []

    def log\_step(self, message):

        """Log computation steps"""

        self.computation\_steps.append(message)

        print(message)

    def compute\_first(self, symbol):

        """Compute FIRST set with memoization and detailed steps"""

        # Check if already computed (memoization)

        if symbol in self.first\_sets:

            self.log\_step(f"✓ FIRST({symbol}) already computed: {self.first\_sets[symbol]}")

            return self.first\_sets[symbol]

        # Check for left recursion

        if symbol in self.first\_in\_progress:

            self.log\_step(f"⚠ Left recursion detected for {symbol}, returning empty set")

            return set()

        self.first\_in\_progress.add(symbol)

        self.log\_step(f"\n🔄 Computing FIRST({symbol})...")

        first = set()

        # Terminal case

        if symbol not in self.grammar:

            first.add(symbol)

            self.log\_step(f"  📝 {symbol} is terminal → FIRST({symbol}) = {{{symbol}}}")

        else:

            # Non-terminal case

            self.log\_step(f"  📝 {symbol} is non-terminal with productions: {self.grammar[symbol]}")

            for i, production in enumerate(self.grammar[symbol]):

                self.log\_step(f"    Production {i+1}: {symbol} → {' '.join(production)}")

                if production == ["ε"]:

                    first.add("ε")

                    self.log\_step(f"      ✅ Added ε to FIRST({symbol})")

                else:

                    # Process each symbol in the production

                    all\_have\_epsilon = True

                    for j, char in enumerate(production):

                        self.log\_step(f"      🔍 Processing symbol {j+1}: {char}")

                        # Recursive call with memoization

                        first\_char = self.compute\_first(char)

                        # Add non-epsilon symbols

                        before\_size = len(first)

                        first |= (first\_char - {"ε"})

                        new\_symbols = first - (first - (first\_char - {"ε"})) if before\_size < len(first) else set()

                        if new\_symbols:

                            self.log\_step(f"        ➕ Added {new\_symbols} from FIRST({char}) to FIRST({symbol})")

                        # Check if epsilon is in FIRST(char)

                        if "ε" not in first\_char:

                            self.log\_step(f"        🛑 {char} doesn't derive ε, stopping here")

                            all\_have\_epsilon = False

                            break

                        else:

                            self.log\_step(f"        ⏭ {char} can derive ε, continuing...")

                    # If all symbols can derive epsilon, add epsilon

                    if all\_have\_epsilon:

                        first.add("ε")

                        self.log\_step(f"      ✅ All symbols derive ε → Added ε to FIRST({symbol})")

        # Store result (memoization)

        self.first\_sets[symbol] = first

        self.first\_in\_progress.remove(symbol)

        self.log\_step(f"  ✅ Final FIRST({symbol}) = {first}")

        return first

    def compute\_follow(self, symbol, start\_symbol):

        """Compute FOLLOW set with memoization and detailed steps"""

        # Initialize if first time

        if symbol not in self.follow\_sets:

            self.follow\_sets[symbol] = set()

        # Check for left recursion

        if symbol in self.follow\_in\_progress:

            self.log\_step(f"⚠ FOLLOW recursion detected for {symbol}, returning current set")

            return self.follow\_sets[symbol]

        self.follow\_in\_progress.add(symbol)

        self.log\_step(f"\n🔄 Computing FOLLOW({symbol})...")

        initial\_set = self.follow\_sets[symbol].copy()

        # Start symbol gets $

        if symbol == start\_symbol and "$" not in self.follow\_sets[symbol]:

            self.follow\_sets[symbol].add("$")

            self.log\_step(f"  📝 {symbol} is start symbol → Added $ to FOLLOW({symbol})")

        # Look for symbol in all productions

        for lhs in self.grammar:

            for prod\_num, production in enumerate(self.grammar[lhs]):

                self.log\_step(f"  🔍 Checking: {lhs} → {' '.join(production)}")

                for i, char in enumerate(production):

                    if char == symbol:

                        self.log\_step(f"    ✓ Found {symbol} at position {i}")

                        # Case 1: There's a symbol after current symbol

                        if i + 1 < len(production):

                            next\_symbol = production[i + 1]

                            self.log\_step(f"      📍 Next symbol: {next\_symbol}")

                            # Get FIRST of next symbol (might trigger computation)

                            if next\_symbol not in self.first\_sets:

                                self.compute\_first(next\_symbol)

                            next\_first = self.first\_sets[next\_symbol]

                            before\_size = len(self.follow\_sets[symbol])

                            self.follow\_sets[symbol] |= (next\_first - {"ε"})

                            if len(self.follow\_sets[symbol]) > before\_size:

                                added = (next\_first - {"ε"})

                                self.log\_step(f"        ➕ Added {added} from FIRST({next\_symbol}) to FOLLOW({symbol})")

                            # If FIRST(next\_symbol) contains ε, add FOLLOW(lhs)

                            if "ε" in next\_first:

                                self.log\_step(f"        ⚠ {next\_symbol} can derive ε → Need FOLLOW({lhs})")

                                if lhs != symbol:  # Avoid infinite recursion

                                    before\_size = len(self.follow\_sets[symbol])

                                    follow\_lhs = self.compute\_follow(lhs, start\_symbol)

                                    self.follow\_sets[symbol] |= follow\_lhs

                                    if len(self.follow\_sets[symbol]) > before\_size:

                                        added = follow\_lhs

                                        self.log\_step(f"        ➕ Added {added} from FOLLOW({lhs}) to FOLLOW({symbol})")

                        # Case 2: Symbol is at the end of production

                        else:

                            self.log\_step(f"      📍 {symbol} is at end of production")

                            if lhs != symbol:  # Avoid infinite recursion

                                self.log\_step(f"        ⚠ Need FOLLOW({lhs})")

                                before\_size = len(self.follow\_sets[symbol])

                                follow\_lhs = self.compute\_follow(lhs, start\_symbol)

                                self.follow\_sets[symbol] |= follow\_lhs

                                if len(self.follow\_sets[symbol]) > before\_size:

                                    added = follow\_lhs

                                    self.log\_step(f"        ➕ Added {added} from FOLLOW({lhs}) to FOLLOW({symbol})")

        self.follow\_in\_progress.remove(symbol)

        # Check if anything was added

        if self.follow\_sets[symbol] != initial\_set:

            self.log\_step(f"  ✅ Updated FOLLOW({symbol}) = {self.follow\_sets[symbol]}")

        else:

            self.log\_step(f"  ✅ No changes to FOLLOW({symbol}) = {self.follow\_sets[symbol]}")

        return self.follow\_sets[symbol]

def main():

    grammar = defaultdict(list)

    filename = "grammar.txt"

    # Read grammar from file

    print("📖 Reading grammar from file...")

    with open(filename, "r") as f:

        for line in f:

            if "->" in line:

                lhs, rhs = line.strip().split("->")

                lhs = lhs.strip()

                productions = rhs.strip().split("|")

                for prod in productions:

                    prod\_symbols = prod.strip().split()

                    prod\_symbols = ["ε" if sym.lower() == "epsilon" else sym for sym in prod\_symbols]

                    grammar[lhs].append(prod\_symbols)

    print(f"📋 Grammar loaded:")

    for nt in grammar:

        for i, prod in enumerate(grammar[nt]):

            print(f"  {nt} → {' '.join(prod)}")

    # Initialize calculator

    calculator = FirstFollowCalculator(grammar)

    start\_symbol = list(grammar.keys())[0]

    # Compute FIRST sets

    print("\n" + "="\*60)

    print("🎯 COMPUTING FIRST SETS WITH MEMOIZATION")

    print("="\*60)

    for non\_terminal in grammar:

        print(f"\n{'='\*20} FIRST({non\_terminal}) {'='\*20}")

        calculator.compute\_first(non\_terminal)

    # Compute FOLLOW sets

    print("\n" + "="\*60)

    print("🎯 COMPUTING FOLLOW SETS WITH MEMOIZATION")

    print("="\*60)

    for non\_terminal in grammar:

        print(f"\n{'='\*20} FOLLOW({non\_terminal}) {'='\*20}")

        calculator.compute\_follow(non\_terminal, start\_symbol)

    # Final results

    print("\n" + "="\*60)

    print("📊 FINAL RESULTS")

    print("="\*60)

    print("\n🔤 FIRST SETS:")

    for nt in grammar:

        print(f"  FIRST({nt}) = {calculator.first\_sets[nt]}")

    print("\n🔤 FOLLOW SETS:")

    for nt in grammar:

        print(f"  FOLLOW({nt}) = {calculator.follow\_sets[nt]}")

    print("\n📈 COMPUTATION STATISTICS:")

    print(f"  Total FIRST computations avoided by memoization: {len([s for s in calculator.computation\_steps if 'already computed' in s])}")

    print(f"  Total computation steps logged: {len(calculator.computation\_steps)}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**grammar.txt:**

E -> T X

X -> + T X | epsilon

T -> F Y

Y -> \* F Y | epsilon

F -> ( E ) | id

**OUTPUT:**

PS C:\Users\Joseph\Desktop\compiler design> cd expt4

PS C:\Users\Joseph\Desktop\compiler design\expt4> python expt4a\_optimized.py

📖 Reading grammar from file...

📋 Grammar loaded:

E → T X

X → + T X

X → ε

T → F Y

Y → \* F Y

Y → ε

F → ( E )

F → id

============================================================

🎯 COMPUTING FIRST SETS WITH MEMOIZATION

============================================================

==================== FIRST(E) ====================

🔄 Computing FIRST(E)...

📝 E is non-terminal with productions: [['T', 'X']]

Production 1: E → T X

🔍 Processing symbol 1: T

🔄 Computing FIRST(T)...

📝 T is non-terminal with productions: [['F', 'Y']]

Production 1: T → F Y

🔍 Processing symbol 1: F

🔄 Computing FIRST(F)...

📝 F is non-terminal with productions: [['(', 'E', ')'], ['id']]

Production 1: F → ( E )

🔍 Processing symbol 1: (

🔄 Computing FIRST(()...

📝 ( is terminal → FIRST(() = {(}

✅ Final FIRST(() = {'('}

➕ Added {'('} from FIRST(() to FIRST(F)

🛑 ( doesn't derive ε, stopping here

Production 2: F → id

🔍 Processing symbol 1: id

🔄 Computing FIRST(id)...

📝 id is terminal → FIRST(id) = {id}

✅ Final FIRST(id) = {'id'}

➕ Added {'id'} from FIRST(id) to FIRST(F)

🛑 id doesn't derive ε, stopping here

✅ Final FIRST(F) = {'(', 'id'}

➕ Added {'(', 'id'} from FIRST(F) to FIRST(T)

🛑 F doesn't derive ε, stopping here

✅ Final FIRST(T) = {'(', 'id'}

➕ Added {'(', 'id'} from FIRST(T) to FIRST(E)

🛑 T doesn't derive ε, stopping here

✅ Final FIRST(E) = {'(', 'id'}

==================== FIRST(X) ====================

🔄 Computing FIRST(X)...

📝 X is non-terminal with productions: [['+', 'T', 'X'], ['ε']]

Production 1: X → + T X

🔍 Processing symbol 1: +

🔄 Computing FIRST(+)...

📝 + is terminal → FIRST(+) = {+}

✅ Final FIRST(+) = {'+'}

➕ Added {'+'} from FIRST(+) to FIRST(X)

🛑 + doesn't derive ε, stopping here

Production 2: X → ε

✅ Added ε to FIRST(X)

✅ Final FIRST(X) = {'+', 'ε'}

==================== FIRST(T) ====================

✓ FIRST(T) already computed: {'(', 'id'}

==================== FIRST(Y) ====================

🔄 Computing FIRST(Y)...

📝 Y is non-terminal with productions: [['\*', 'F', 'Y'], ['ε']]

Production 1: Y → \* F Y

🔍 Processing symbol 1: \*

🔄 Computing FIRST(\*)...

📝 \* is terminal → FIRST(\*) = {\*}

✅ Final FIRST(\*) = {'\*'}

➕ Added {'\*'} from FIRST(\*) to FIRST(Y)

🛑 \* doesn't derive ε, stopping here

Production 2: Y → ε

✅ Added ε to FIRST(Y)

✅ Final FIRST(Y) = {'\*', 'ε'}

==================== FIRST(F) ====================

✓ FIRST(F) already computed: {'(', 'id'}

============================================================

🎯 COMPUTING FOLLOW SETS WITH MEMOIZATION

============================================================

==================== FOLLOW(E) ====================

🔄 Computing FOLLOW(E)...

📝 E is start symbol → Added $ to FOLLOW(E)

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

✓ Found E at position 1

📍 Next symbol: )

🔄 Computing FIRST())...

📝 ) is terminal → FIRST()) = {)}

✅ Final FIRST()) = {')'}

➕ Added {')'} from FIRST()) to FOLLOW(E)

🔍 Checking: F → id

✅ Updated FOLLOW(E) = {'$', ')'}

==================== FOLLOW(X) ====================

🔄 Computing FOLLOW(X)...

🔍 Checking: E → T X

✓ Found X at position 1

📍 X is at end of production

⚠ Need FOLLOW(E)

🔄 Computing FOLLOW(E)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

✓ Found E at position 1

📍 Next symbol: )

🔍 Checking: F → id

✅ No changes to FOLLOW(E) = {'$', ')'}

➕ Added {'$', ')'} from FOLLOW(E) to FOLLOW(X)

🔍 Checking: X → + T X

✓ Found X at position 2

📍 X is at end of production

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ Updated FOLLOW(X) = {'$', ')'}

==================== FOLLOW(T) ====================

🔄 Computing FOLLOW(T)...

🔍 Checking: E → T X

✓ Found T at position 0

📍 Next symbol: X

➕ Added {'+'} from FIRST(X) to FOLLOW(T)

⚠ X can derive ε → Need FOLLOW(E)

🔄 Computing FOLLOW(E)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

✓ Found E at position 1

📍 Next symbol: )

🔍 Checking: F → id

✅ No changes to FOLLOW(E) = {'$', ')'}

➕ Added {'$', ')'} from FOLLOW(E) to FOLLOW(T)

🔍 Checking: X → + T X

✓ Found T at position 1

📍 Next symbol: X

⚠ X can derive ε → Need FOLLOW(X)

🔄 Computing FOLLOW(X)...

🔍 Checking: E → T X

✓ Found X at position 1

📍 X is at end of production

⚠ Need FOLLOW(E)

🔄 Computing FOLLOW(E)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

✓ Found E at position 1

📍 Next symbol: )

🔍 Checking: F → id

✅ No changes to FOLLOW(E) = {'$', ')'}

🔍 Checking: X → + T X

✓ Found X at position 2

📍 X is at end of production

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ No changes to FOLLOW(X) = {'$', ')'}

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ Updated FOLLOW(T) = {'$', '+', ')'}

==================== FOLLOW(Y) ====================

🔄 Computing FOLLOW(Y)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

✓ Found Y at position 1

📍 Y is at end of production

⚠ Need FOLLOW(T)

🔄 Computing FOLLOW(T)...

🔍 Checking: E → T X

✓ Found T at position 0

📍 Next symbol: X

⚠ X can derive ε → Need FOLLOW(E)

🔄 Computing FOLLOW(E)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

✓ Found E at position 1

📍 Next symbol: )

🔍 Checking: F → id

✅ No changes to FOLLOW(E) = {'$', ')'}

🔍 Checking: X → + T X

✓ Found T at position 1

📍 Next symbol: X

⚠ X can derive ε → Need FOLLOW(X)

🔄 Computing FOLLOW(X)...

🔍 Checking: E → T X

✓ Found X at position 1

📍 X is at end of production

⚠ Need FOLLOW(E)

🔄 Computing FOLLOW(E)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

✓ Found E at position 1

📍 Next symbol: )

🔍 Checking: F → id

✅ No changes to FOLLOW(E) = {'$', ')'}

🔍 Checking: X → + T X

✓ Found X at position 2

📍 X is at end of production

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ No changes to FOLLOW(X) = {'$', ')'}

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ No changes to FOLLOW(T) = {'$', '+', ')'}

➕ Added {'$', '+', ')'} from FOLLOW(T) to FOLLOW(Y)

🔍 Checking: Y → \* F Y

✓ Found Y at position 2

📍 Y is at end of production

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ Updated FOLLOW(Y) = {'$', '+', ')'}

==================== FOLLOW(F) ====================

🔄 Computing FOLLOW(F)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

✓ Found F at position 0

📍 Next symbol: Y

➕ Added {'\*'} from FIRST(Y) to FOLLOW(F)

⚠ Y can derive ε → Need FOLLOW(T)

🔄 Computing FOLLOW(T)...

🔍 Checking: E → T X

✓ Found T at position 0

📍 Next symbol: X

⚠ X can derive ε → Need FOLLOW(E)

🔄 Computing FOLLOW(E)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

✓ Found E at position 1

📍 Next symbol: )

🔍 Checking: F → id

✅ No changes to FOLLOW(E) = {'$', ')'}

🔍 Checking: X → + T X

✓ Found T at position 1

📍 Next symbol: X

⚠ X can derive ε → Need FOLLOW(X)

🔄 Computing FOLLOW(X)...

🔍 Checking: E → T X

✓ Found X at position 1

📍 X is at end of production

⚠ Need FOLLOW(E)

🔄 Computing FOLLOW(E)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

✓ Found E at position 1

📍 Next symbol: )

🔍 Checking: F → id

✅ No changes to FOLLOW(E) = {'$', ')'}

🔍 Checking: X → + T X

✓ Found X at position 2

📍 X is at end of production

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ No changes to FOLLOW(X) = {'$', ')'}

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ No changes to FOLLOW(T) = {'$', '+', ')'}

➕ Added {'$', '+', ')'} from FOLLOW(T) to FOLLOW(F)

🔍 Checking: Y → \* F Y

✓ Found F at position 1

📍 Next symbol: Y

⚠ Y can derive ε → Need FOLLOW(Y)

🔄 Computing FOLLOW(Y)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

✓ Found Y at position 1

📍 Y is at end of production

⚠ Need FOLLOW(T)

🔄 Computing FOLLOW(T)...

🔍 Checking: E → T X

✓ Found T at position 0

📍 Next symbol: X

⚠ X can derive ε → Need FOLLOW(E)

🔄 Computing FOLLOW(E)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

✓ Found E at position 1

📍 Next symbol: )

🔍 Checking: F → id

✅ No changes to FOLLOW(E) = {'$', ')'}

🔍 Checking: X → + T X

✓ Found T at position 1

📍 Next symbol: X

⚠ X can derive ε → Need FOLLOW(X)

🔄 Computing FOLLOW(X)...

🔍 Checking: E → T X

✓ Found X at position 1

📍 X is at end of production

⚠ Need FOLLOW(E)

🔄 Computing FOLLOW(E)...

🔍 Checking: E → T X

🔍 Checking: X → + T X

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

✓ Found E at position 1

📍 Next symbol: )

🔍 Checking: F → id

✅ No changes to FOLLOW(E) = {'$', ')'}

🔍 Checking: X → + T X

✓ Found X at position 2

📍 X is at end of production

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ No changes to FOLLOW(X) = {'$', ')'}

🔍 Checking: X → ε

🔍 Checking: T → F Y

🔍 Checking: Y → \* F Y

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ No changes to FOLLOW(T) = {'$', '+', ')'}

🔍 Checking: Y → \* F Y

✓ Found Y at position 2

📍 Y is at end of production

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ No changes to FOLLOW(Y) = {'$', '+', ')'}

🔍 Checking: Y → ε

🔍 Checking: F → ( E )

🔍 Checking: F → id

✅ Updated FOLLOW(F) = {'$', '+', ')', '\*'}

============================================================

📊 FINAL RESULTS

============================================================

🔤 FIRST SETS:

FIRST(E) = {'(', 'id'}

FIRST(X) = {'+', 'ε'}

FIRST(T) = {'(', 'id'}

FIRST(Y) = {'\*', 'ε'}

FIRST(F) = {'(', 'id'}

🔤 FOLLOW SETS:

FOLLOW(E) = {'$', ')'}

FOLLOW(X) = {'$', ')'}

FOLLOW(T) = {'$', '+', ')'}

FOLLOW(Y) = {'$', '+', ')'}

FOLLOW(F) = {'$', '+', ')', '\*'}

📈 COMPUTATION STATISTICS:

Total FIRST computations avoided by memoization: 2

Total computation steps logged: 373

PS C:\Users\Joseph\Desktop\compiler design\expt4>

**CONCLUSION:**

We successfully implemented FIRST and FOLLOW set computation algorithms for context-free grammars, demonstrating essential concepts in parser construction. The program correctly computed sets for an arithmetic expression grammar, properly handling epsilon productions and recursive dependencies. Key learnings include understanding how FIRST sets determine valid starting tokens, how FOLLOW sets identify lookahead symbols, and how these algorithms form the foundation for LL (1) predictive parsing. This experiment bridges theoretical compiler concepts with practical implementation, providing crucial knowledge for syntax analysis and parser design.