PROGRAM:

```
import java.util.*;
public class CNFConverter {
   private static final int MAX_LETTER = 26;
   public static void main(String[] args) {
         Scanner scanner = new Scanner(System.in);
         System.out.print("Enter the NonTerminal Symbols: ");
         String[] nonTerminalSymbols = scanner.nextLine().split(" ");
        System.out.print("Enter the Terminal Symbols: ");
        String[] terminalSymbols = scanner.nextLine().split(" ");
        Map<String, List<String>> productions = getProductions(nonTerminalSymbols, scanner);
        eliminateNullProductions(productions);
        removeUnitProductions(productions);
        System.out.println("\nAFTER REMOVING NULL AND UNIT PRODUCTION");
        printProductions(productions);
        System.out.println("\nConverting to CNF form: ");
        convertToCNF(productions, nonTerminalSymbols);
        System.out.println("\nFINAL CNF PRODUCTIONS:");
       printProductions(productions);
     }
 private static Map<String, List<String>> getProductions(String[] nonTerminalSymbols, Scanner scanner) {
      Map<String, List<String>> productions = new HashMap<>();
 for (String symbol : nonTerminalSymbols) {
         System.out.print(symbol + " -> ");
         String[] productionRules = scanner.nextLine().split("/");
         productions.put(symbol, new ArrayList<>(Arrays.asList(productionRules)));
      }
      return productions;
      }
```

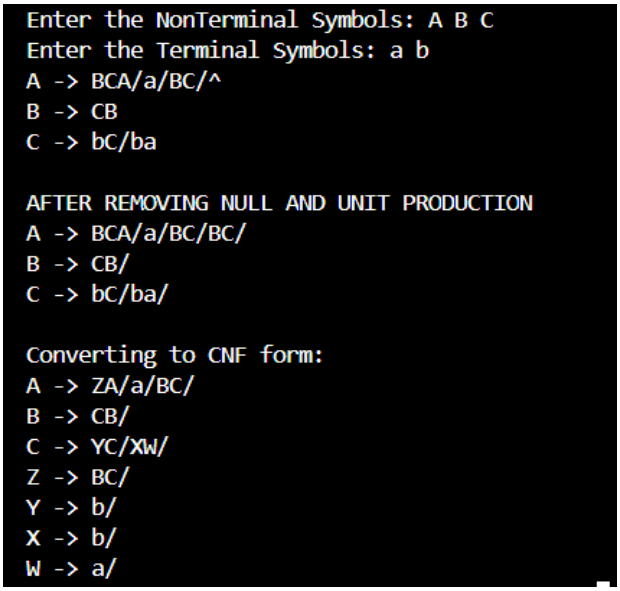
```
private static void eliminateNullProductions(Map<String, List<String>> productions) {
      Set<String> nullableNonTerminals = findNullableNonTerminals(productions);
      for (List<String> productionRules : productions.values()) {
         productionRules.removeIf(rule -> rule.equals("^"));
      }
      for (Map.Entry<String, List<String>> entry : productions.entrySet()) {
         String symbol = entry.getKey();
         List<String> productionRules = entry.getValue();
         List<String> newRules = new ArrayList<>();  // Collect new rules here
         for (String rule : productionRules) {
            if (rule.length() > 1) {
               for (int i = 0; i < rule.length(); i++) {
                  if (nullableNonTerminals.contains(String.valueOf(rule.charAt(i)))) {
                     String newRule = rule.substring(0, i) + rule.substring(i + 1);
                     newRules.add(newRule);  // Add new rules to this list
                  }
               }
            }
         }
         productionRules.addAll(newRules);  // Add all new rules after iteration
      }
   }
private static Set<String> findNullableNonTerminals(Map<String, List<String>> productions) {
       Set<String> nullableNonTerminals = new HashSet<>();
      for (Map.Entry<String, List<String>> entry : productions.entrySet()) {
            String symbol = entry.getKey();
            List<String> productionRules = entry.getValue();
         for (String rule : productionRules) {
               if (rule.equals("^")) {
                nullableNonTerminals.add(symbol);
```

```
            }
        }
    }
    return nullableNonTerminals;
}
private static void removeUnitProductions(Map<String, List<String>> productions) {
    Map<String, List<String>> unitProductions = findUnitProductions(productions);
    for (Map.Entry<String, List<String>> entry : unitProductions.entrySet()) {
            String symbol = entry.getKey();
            List<String> productionRules = entry.getValue();
        for (String rule : productionRules) {
            productions.get(symbol).remove(rule);
            productions.get(symbol).addAll(productions.get(rule));
        }
    }
}
private static Map<String, List<String>> findUnitProductions(Map<String, List<String>> productions) {
    Map<String, List<String>> unitProductions = new HashMap<>();
  for (Map.Entry<String, List<String>> entry : productions.entrySet()) {
        String symbol = entry.getKey();
        List<String> productionRules = entry.getValue();
    for (String rule : productionRules) {
            if (productions.containsKey(rule) && !symbol.equals(rule)) {
                unitProductions.computeIfAbsent(symbol, k -> new ArrayList<>()).add(rule);
            }
        }
    }
    return unitProductions;
}
```

```
    private static void convertToCNF(Map<String, List<String>> productions, String[] nonTerminalSymbols)
{

    }
private static void printProductions(Map<String, List<String>> productions) {
        for (Map.Entry<String, List<String>> entry : productions.entrySet()) {
            String symbol = entry.getKey();
            List<String> productionRules = entry.getValue();
            System.out.println(symbol + " -> " + String.join("/", productionRules));
        }
    }
}
```

OUTPUT:



OBSERVATION:

In this experiment, I learned about how to eliminate null and unit production stepwise by applying rules and following sequential steps, how we can then convert into Chomsky Normal Form(CNF). For unit production finding derivables and then replace with its own production .

CONCLUSION:

Through this experiment, I gained a deeper understanding of how to eliminate null and unit production from the given grammar. Also I can identify nullable variables and derivables by applying rules. After eliminating null and unit production, we have to convert into standard Chomsky Normal Form.