2.C)Implementation of Backward Chaining

AIM:

To implement Backward Chaining to determine whether a goal (query) can be inferred from a set of known facts and rules.

CODE:

```
def backward_chaining(rules, facts, goal):
    inferred_facts = set(facts)
    agenda = [goal]
    path = {}

    while agenda:
        current_goal = agenda.pop(0)

        if current_goal in inferred_facts:
            continue

        for rule in rules:
            if rule['consequent'] == current_goal:
                all_premises_true = True
                for premise in rule['antecedents']:
                    if premise not in inferred_facts:
                        agenda.append(premise)
                        path[premise] = current_goal
                        all_premises_true = False
                        break

                if all_premises_true:
                    inferred_facts.add(current_goal)
                    print(f"Inferred: {current_goal}")
                    break
```

```
    return goal in inferred_facts


def construct_path(path, goal):
    current = goal
    full_path = [goal]
    while current in path:
      current = path[current]
      full_path.append(current)
    return full_path[::-1]

rules = [
    {'antecedents': ['A', 'B'], 'consequent': 'C'},
    {'antecedents': ['C', 'D'], 'consequent': 'E'},
    {'antecedents': ['F'], 'consequent': 'D'},
    {'antecedents': ['G'], 'consequent': 'A'},
]

facts = ['A','F','B','G']
goal = 'E'

if backward_chaining(rules, facts, goal):
    print(f"Goal '{goal}' can be proven.")
else:
    print(f"Goal '{goal}' cannot be proven.")
```

OUTPUT:

Inferred: C
Goal 'E' cannot be proven.

RESULT:

The code is executed as expected and the output have been verified successfully.