In [1]:

```
import sys

#Some lists required for algorithm
start_state = [] #required for storing initial state
goal_state = []  #required for storing final state
current_state = [] #required for storing current ongoing state
planning_stack = [] #stack required in goal stack planning (only going to add sub goal
s)
actual_plan = []  #plan generated (output)

#actions and predicates
actions = ["stack", "unstack", "pickup", "putdown"]
predicates = ["on", "clear", "arm_empty", "holding", "on_table"]

#necessary functions required for algorithm

#Preconditions append functions:-
def preconditions_stack(X, Y):
    planning_stack.append("holding "+str(X))
    planning_stack.append("clear "+str(Y))

def preconditions_unstack(X, Y):
    planning_stack.append("on "+str(X)+" "+str(Y))
    planning_stack.append("clear "+str(X))

def preconditions_pickup(X):
    planning_stack.append("arm_empty")
    planning_stack.append("on_table "+str(X))
    planning_stack.append("clear "+str(X))


def preconditions_putdown(X):
    planning_stack.append("holding "+str(X))


#Corresponding action required to satisfy the predicates
def for_on(X, Y):
    planning_stack.append("stack "+str(X)+" "+str(Y))
    preconditions_stack(X, Y)

def for_ontable(X):
    planning_stack.append("putdown "+str(X))
    preconditions_putdown(X)

def for_clear(X):
    #Finding the block on which X is stacked
    check = "on "

    for predicate in current_state:
        if check in predicate:
            temp_list = predicate.split()

            if temp_list[2] == X:
                break

    Y = str(temp_list[1])

    #Appending Unstack operatrion
    planning_stack.append("unstack "+str(Y)+" "+str(X))
```

```
    preconditions_unstack(Y, X)

def for_holding(X):
    check = "on_table "+str(X)

    if check in current_state:
        planning_stack.append("pickup "+str(X))
        preconditions_pickup(X)

    else:
         #Finding the block on which X is stacked
        check = "on "

        for predicate in current_state:
            if check in predicate:
                temp_list = predicate.split()

                if temp_list[1] == X:
                    break

        Y = str(temp_list[2])

        #Appending Unstack operatrion
        planning_stack.append("unstack "+str(X)+" "+str(Y))
        preconditions_unstack(X, Y)

def for_armempty():
    print("\nArm Empty error\n")
    sys.quit()




#Effects of action
def effect_stack(X, Y):
    current_state.remove("holding "+str(X))
    current_state.remove("clear "+str(Y))

    current_state.append("on "+str(X)+" "+str(Y))
    current_state.append("clear "+str(X))
    current_state.append("arm_empty")

def effect_unstack(X, Y):
    current_state.remove("on "+str(X)+" "+str(Y))
    current_state.remove("clear "+str(X))
    current_state.remove("arm_empty")

    current_state.append("holding "+str(X))
    current_state.append("clear "+str(Y))

def effect_pickup(X):
    current_state.remove("arm_empty")
    current_state.remove("on_table "+str(X))
    current_state.remove("clear "+str(X))

    current_state.append("holding "+str(X))

def effect_putdown(X):
    current_state.remove("holding "+str(X))
```

```
    current_state.append("arm_empty")
    current_state.append("on_table "+str(X))
    current_state.append("clear "+str(X))
```

Actual Algorithm

while stack is not empty:
  if top of stack is predicate:
    if predicate is true:
      pop it
    else:
      pop it
      push corresponding action that will satisfy that predicate onto stack
      push preconditions of that action
  if top of stack is action:
    pop it
    perform the action i.e add and delete it's effects from current state.
    add that action to the actual plan

In [2]:

```
input_string = input("Enter start state:- ")
start_state = input_string.split("^")

input_string = input("Enter goal state:- ")
goal_state = input_string.split("^")

print("\nEntered Start State:- "+str(start_state))
print("\nEntered Goal State:- "+str(goal_state)+"\n")

current_state = start_state.copy()

for predicate in goal_state:
    planning_stack.append(predicate)

while len(planning_stack) > 0:
    print("Planning Stack:- "+str(planning_stack))
    print("Current State:- "+str(current_state)+"\n")

    top = planning_stack.pop()
    temp = top.split()

    if temp[0] in predicates: #if top of stack is predicate

        if top in current_state: #if predicate is true:
            continue #You have already poped it.

        else:
            #Already poped above

            #push corresponding action that will satisfy that predicate onto stack and
 push preconditions of that action
            if temp[0] == "on":
                for_on(temp[1], temp[2])
            elif temp[0] == "on_table":
                for_ontable(temp[1])
            elif temp[0] == "clear":
                for_clear(temp[1])
            elif temp[0] == "holding":
                for_holding(temp[1])
            elif temp[0] == "arm_empty":
                for_armempty()

    if temp[0] in actions: #if top of stack is action
        #Already poped above

        #perform the action i.e add and delete it's effects from current state
        if temp[0] == "stack":
            effect_stack(temp[1], temp[2])
        elif temp[0] == "unstack":
            effect_unstack(temp[1], temp[2])
        elif temp[0] == "pickup":
            effect_pickup(temp[1])
        elif temp[0] == "putdown":
            effect_putdown(temp[1])

        #add that action to the actual plan
        actual_plan.append(top)
```

```
print("Final Current State:- "+str(current_state))

print("\nPlan Generated:- \n")
for step in actual_plan:
    print(step)
```

```
Enter start state:- on B A^on_table A^on_table C^clear B^clear C^arm_empty
Enter goal state:- on A B^on B C^on_table C^clear A^arm_empty

Entered Start State:- ['on B A', 'on_table A', 'on_table C', 'clear B', 'c
lear C', 'arm_empty']

Entered Goal State:- ['on A B', 'on B C', 'on_table C', 'clear A', 'arm_em
pty']

Planning Stack:- ['on A B', 'on B C', 'on_table C', 'clear A', 'arm_empt
y']
Current State:- ['on B A', 'on_table A', 'on_table C', 'clear B', 'clear
C', 'arm_empty']

Planning Stack:- ['on A B', 'on B C', 'on_table C', 'clear A']
Current State:- ['on B A', 'on_table A', 'on_table C', 'clear B', 'clear
C', 'arm_empty']

Planning Stack:- ['on A B', 'on B C', 'on_table C', 'unstack B A', 'on B
A', 'clear B']
Current State:- ['on B A', 'on_table A', 'on_table C', 'clear B', 'clear
C', 'arm_empty']

Planning Stack:- ['on A B', 'on B C', 'on_table C', 'unstack B A', 'on B
A']
Current State:- ['on B A', 'on_table A', 'on_table C', 'clear B', 'clear
C', 'arm_empty']

Planning Stack:- ['on A B', 'on B C', 'on_table C', 'unstack B A']
Current State:- ['on B A', 'on_table A', 'on_table C', 'clear B', 'clear
C', 'arm_empty']

Planning Stack:- ['on A B', 'on B C', 'on_table C']
Current State:- ['on_table A', 'on_table C', 'clear C', 'holding B', 'clea
r A']

Planning Stack:- ['on A B', 'on B C']
Current State:- ['on_table A', 'on_table C', 'clear C', 'holding B', 'clea
r A']

Planning Stack:- ['on A B', 'stack B C', 'holding B', 'clear C']
Current State:- ['on_table A', 'on_table C', 'clear C', 'holding B', 'clea
r A']

Planning Stack:- ['on A B', 'stack B C', 'holding B']
Current State:- ['on_table A', 'on_table C', 'clear C', 'holding B', 'clea
r A']

Planning Stack:- ['on A B', 'stack B C']
Current State:- ['on_table A', 'on_table C', 'clear C', 'holding B', 'clea
r A']

Planning Stack:- ['on A B']
Current State:- ['on_table A', 'on_table C', 'clear A', 'on B C', 'clear
B', 'arm_empty']

Planning Stack:- ['stack A B', 'holding A', 'clear B']
Current State:- ['on_table A', 'on_table C', 'clear A', 'on B C', 'clear
B', 'arm_empty']

Planning Stack:- ['stack A B', 'holding A']
```

```
Current State:- ['on_table A', 'on_table C', 'clear A', 'on B C', 'clear
B', 'arm_empty']

Planning Stack:- ['stack A B', 'pickup A', 'arm_empty', 'on_table A', 'cle
ar A']
Current State:- ['on_table A', 'on_table C', 'clear A', 'on B C', 'clear
B', 'arm_empty']

Planning Stack:- ['stack A B', 'pickup A', 'arm_empty', 'on_table A']
Current State:- ['on_table A', 'on_table C', 'clear A', 'on B C', 'clear
B', 'arm_empty']

Planning Stack:- ['stack A B', 'pickup A', 'arm_empty']
Current State:- ['on_table A', 'on_table C', 'clear A', 'on B C', 'clear
B', 'arm_empty']

Planning Stack:- ['stack A B', 'pickup A']
Current State:- ['on_table A', 'on_table C', 'clear A', 'on B C', 'clear
B', 'arm_empty']

Planning Stack:- ['stack A B']
Current State:- ['on_table C', 'on B C', 'clear B', 'holding A']

Final Current State:- ['on_table C', 'on B C', 'on A B', 'clear A', 'arm_e
mpty']

Plan Generated:-

unstack B A
stack B C
pickup A
stack A B
```

In [3]:

```
#trial
#on B A^on_table A^clear B^arm_empty
#on A B^on_table B^clear A^arm_empty

#ques 1
#on B A^on_table A^on_table C^on_table D^clear B^clear C^clear D^arm_empty
#on C A^on B D^on_table A^on_table D^clear C^clear B^arm_empty

#ques 2
#on C A^on B D^on_table A^on_table D^clear C^clear B^arm_empty
#on A B^on B D^on_table D^on_table C^clear A^clear C^arm_empty

#ques 3
#on B A^on_table A^on_table C^clear B^clear C^arm_empty
#on A B^on B C^on_table C^clear A^arm_empty
```

In [ ]:

```

```