Prolog

MCS-705(b)

List of Programs

Q.1 WAP for Addition, Subtraction, Multiplication and Division of two numbers.

Program:-

```
add(X,Y):- Z is X+Y,write(X),write(' + '),write(Y),
 write(' = '),write(Z).
sub(X,Y):- Z is X-Y,write(X),write(' - '),write(Y),
 write(' = '),write(Z).
mul(X,Y):- Z is X*Y,write(X),write(' * '),write(Y),
 write(' = '),write(Z).
div(_,N):- N=0, write('Division by zero is not permitted!'),nl,!,fail.
div(X,Y):- Z is X/Y,write(X),write(' / '),write(Y),
 write(' = '),write(Z).
```

Output:-

```
For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- [q1].
% q1 compiled 0.00 sec, 6 clauses
true.

2 ?- add(56,8).
56 + 8 = 64
true.

 3 ?- sub(13,6) .
13 - 6 = 7
true.

 4 ?- mul(3,35).
3 * 35 = 105
true.

 5 ?- div(22,2).
22 / 2 = 11
true.

  6 ?- div(3,0).
Division by zero is not permitted!
false.
```

Q.2 WAP to compute the factorial of a non-negative number.

Program:-

```
fact:- write('Enter value:' ),nl,read(N),fact(N,F),write('The value of Factorial '),
write(N ),write(' is '),write(F),nl.
fact(0,1).
fact(N,_):- N<0,write('Enter a positive number for its factorial calculation!'),!,fail.
fact(N,F):- N>0, N1 is N-1,fact(N1,F1),F is N*F1.
```

Output:-

```
17 ?- [q2].
% q2 compiled 0.00 sec, 5 clauses
true.

18 ?- fact.
Enter value:
|: 7.
The value of Factorial 7 is 5040
true .

19 ?- fact.
Enter value:
|: 8.
The value of Factorial 8 is 40320
true .

20 ?- fact.
Enter value:
|: 9.
The value of Factorial 9 is 362880
true .

21 ?- fact.
Enter value:
|: -5.
Enter a positive number for its factorial calculation!
false.
```

Q.3 WAP to generate n^{th} Fibonacci numbers.

Program:-

```
fiboseries(N):-
        write('Fibonacci seires: '),
        write('0,1, '),
        fib(N,_,_).
```

```
fib(1, 1, 0).
fib(X, Y1, Y2) :-
X > 1,
X1 is X - 1,
fib(X1, Y2, Y3),
Y1 is Y2 + Y3,
write(Y1),write(', ').
```

Output:-

```
28 ?- [q4].
% q4 compiled 0.00 sec, 4 clauses
true.

29 ?- fiboseries(5).
Fibonacci seires: 0,1, 1, 2, 3, 5,
true
Unknown action: ▪ (h for help)
Action?
Unknown action: ▪ (h for help)
Action? .

30 ?- fiboseries(8).
Fibonacci seires: 0,1, 1, 2, 3, 5, 8, 13, 21,
true .

31 ?- fiboseries(10).
Fibonacci seires: 0,1, 1, 2, 3, 5, 8, 13, 21, 34, 55,
true .
```

Q.4 WAP to solve Towers of Hanoi problem.

Program:-

```
move(1,X,Y,_):- write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),nl.
move(N,X,Y,Z):- N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).
```

Output:-

```
41 ?- [q5].
% q5 compiled 0.00 sec, 3 clauses
true.
```

42 ?- move(2,left_peg,right_peg, centre_peg).
Move top disk from left_peg to centre_peg
Move top disk from left_peg to right_peg
Move top disk from centre_peg to right_peg
true .

43 ?- move(3,left_peg,right_peg, centre_peg).
Move top disk from left_peg to right_peg
Move top disk from left_peg to centre_peg
Move top disk from right_peg to centre_peg
Move top disk from left_peg to right_peg
Move top disk from centre_peg to left_peg
Move top disk from centre_peg to right_peg
Move top disk from left_peg to right_peg
true .

Q. 5 WAP to check whether a given list is palindrome or not?

Program

```
palin(List) :-
reverse(List,List).
reverse(L1,L2) :-
rev(L1, [], L2),
write('This is palindrome.').
rev([],L,L).
rev([H|L],L2,L3) :-
rev(L,[H|L2],L3).
```

Output:-

13 ?- [q5].
% q5 compiled 0.00 sec, 5 clauses
true.

14 ?- palin([a,b,c,d,c,b,a]).
This is palindrome.
true.

15 ?- palin([3,4,5,4,3]).
This is palindrome.
true.

16 ?- palin([a,s,d,f,a]).
false.

17 ?-

Q. 6 Given the following facts-

Steve only likes easy course.

Science course are hard.
All the courses in the basket weaving department are easy.
BK301 is a basket weaving course.

WAP to find “What course would Steve like?”

Program:-

```
likes(steve,X):- easy(X).
hard(X):- science(X).
easy(X):- baskweav(X).
baskweav(bk301).
```

Output:-

```
% q6 compiled 0.00 sec, 5 clauses
true.
47 ?- likes(steve,X).
X = bk301.
48 ?-
```

Q.7 Given the following facts-

All people who are not poor and are smart are happy.
Those people who read are not stupid but smart.
John could read and is wealthy.
Happy people have exciting lives.

WAP to prove “John has an exciting life”.

Program:-

```
happy(X):-wealthy(X),smart(X).
smart(X):-reads(X).
wealthy(john).
reads(john).
exciting_life(A):-happy(A).
```

Output:-

```
51 ?- [q7].
% q7 compiled 0.00 sec, 6 clauses
true.

52 ?- exciting_life(john).
true.
```

Q. 8 Given the following facts-

John likes all food.
Apples are food.
Chicken is food.
Anything anyone eats & isn't killed by is food.
Bill eats Peanuts and is still alive.
Sue eats everything Bill eats.

WAP to prove that "John likes peanuts" and find "what food does Sue eat?"

Program:-

```
likes(john,Y):- food(Y).
food(apple).
food(chicken).
food(Y):- eat(X,Y),alive(X,Y).
alive(bill,peanut).
eat(bill,peanut).
eat(sue,Y):- eat(bill,Y).
```

Output:-

```
1 ?- [q8].
% q8 compiled 0.00 sec, 8 clauses
true.

2 ?- likes(john,peanut).
true .

3 ?- eat(sue,X).
X = peanut.
```

Q.9 Given the following facts-

Marcus was a man.
Marcus was a Pompeian.
All Pompeian's were Roman.
Caesar was a ruler.
All Roman were either loyal to Caesar or hated him.
Everyone is loyal to someone.
People only try to assassinate ruler they are not loyal to.
Marcus try to assassinate Caesar.
All men are people.

WAP to find "Is Marcus loyal to Caesar?", "Does Marcus hate Caesar?"

Program:-

```
man(marcus).
pompeian(marcus).
roman(X):- pompeian(X).
ruler(caesar).
hate(X,caesar):- roman(X),not(loyalto(X,caesar)).
loyalto(X,Y):- man(X),man(Y).
loyalto(X,Y):- person(X),ruler(Y),not(tryassasinate(X,Y)).
tryassasinate(marcus,caesar).
person(X):- man(X).
```

Output:-

```
11 ?- [q10].
% q10 compiled 0.00 sec, 10 clauses
true.

12 ?- loyalto(marcus,caesar).
false.

13 ?- hate(marcus,caesar).
true.
```

Q.10 Given the following facts-

Anyone passing his history examination & winning the lottery is happy.
Anyone who studies or is lucky can pass all his exam.
John didn't study.
John is lucky.
Anyone who is lucky wins the lottery.

WAP to prove "John is happy"

Program:-

```
happy(X):-pass(X,history),win(X,lottery).
pass(X,_):-study(X);lucky(X).
lucky(john).
study(not(john)).
win(X,lottery):-lucky(X).
```

Output:-

```
14 ?- [q11].
% q11 compiled 0.00 sec, 6 clauses
```

true.

15 ?- happy(A).
A = john.

16 ?- happy(john).
true.

Q.11 Given the following knowledge base:

∀x: ∀y: cat(x) Λ fish(y) -> likes-to-eat (x, y)
∀x: calico(x) -> cat(x)
∀x: tuna(y) -> fish(y)
tuna(Charlie)
tuna(Herb)
calico(Pussy)

WAP to represent the knowledge base in prolog implementation and determine "what does pussy likes to eat?"

Program:-

```
likes_to_eat(X,Y):-cat(X),fish(Y).
cat(X):-calico(X).
fish(Y):- tuna(Y).
tuna(charlie).
tuna(herb).
calico(pussy).
```

Output:-

17 ?- [q12].
% q12 compiled 0.00 sec, 7 clauses
true.

18 ?- likes_to_eat(pussy,X).
X = charlie ;
X = herb.

Q.12 Given the following facts-

Sam likes all Indian mild food.
Sam likes all Chinese food.
Sam likes all Italian food.
Sam likes chips.

Curry, dal, tandoori, kurma are Indian food.
Dal, tandoori, kurma are Indian mild food.
chowmein, chopsuey, sweetandsour are Chinese food.
pizza and spaghetti are Italian food.

WAP to find: a) What foods does Sam like?, b) Does Sam like Curry?, c) Does Sam like Chips?

Program:-

```
likes(sam,Food):- indian(Food),mild(Food).
likes(sam,Food):- chinese(Food).
likes(sam,Food):- italian(Food).
likes(sam,chips).
indian(curry).
indian(dahl).
indian(tandoori).
indian(kurma).

mild(dahl).
mild(tandoori).
mild(kurma).
chinese(chow_mein).
chinese(chop_suey).
chinese(sweet_and_sour).

italian(pizza).
italian(spaghetti).
```

Output:-

```
1 ?- [q13].
% q13 compiled 0.00 sec, 17 clauses
true.

2 ?- likes(sam,Food).
Food = dahl ;
Food = tandoori ;
Food = kurma ;
Food = chow_mein ;
Food = chop_suey ;
Food = sweet_and_sour ;
Food = pizza ;
Food = spaghetti ;
Food = chips.

3 ?- likes(sam,curry).
false.

4 ?- likes(sam,chips).
```

true.

Q.13 WAP to concatenate two lists.

Program:-

```
concate([],L1,L1).
   concate([X|Tail],L2,[X|Tail1]):-
      concate(Tail,L2,Tail1).
```

Output:-

```
5 ?- concate([1,2,4,5], [34,6,7],L).
L = [1, 2, 4, 5, 34, 6, 7].

6 ?- concate([1,2,4,5], [4,2,6,7],L).
L = [1, 2, 4, 5, 4, 2, 6, 7].
```

Q.14 WAP to generate first 10 prime numbers and store them in a list**.**

Program:-

```
list:-
write('The list of First 10 prime numbers is: '),
mak(1,0,[]).

add(X,L,[X|L]).

mak(X,Y,List):-
Y < 10 ,
prime(X),
Z is X,
add(X,List,L),
X1 is X+1,
Y1 is Y+1,
mak(X1,Y1,L).

mak(X,Y,List):-
Y < 10 ,
not(prime(X)),
X1 is X+1,
mak(X1,Y,List).

mak(X,Y,List):-
Y = 10,
write(List),nl.
```

```
prime(1).
prime(X):-
check(X,2).


check(X,Y):-
X > Y,
A is X mod Y,
A > 0,
Y1 is Y+1,
check(X,Y1).

check(X,Y):-
X = Y.
```

Output:-

```
3 ?- [q14].
% plin compiled 0.00 sec, 1 clauses
true.

4 ?- list.

The list of First 10 prime numbers is: [23,19,17,13,11,7,5,3,2,1]
true
```

Q.15 WAP for Pre-order, In-order and Post-order traversal of binary trees.

Program:-

```
tree(M):-M=tree(5,tree(4,tree(2,nil,nil),tree(10,nil,nil)),tree(3,nil,nil)).
preorder(tree(X,L,R),Xs) :-
    preorder(L,Ls),
    preorder(R,Rs),
    append([X|Ls],Rs,Xs).
    preorder(_,[]).

inorder(nil, []).
inorder(tree(X, Left, Right), R) :-
   inorder(Left,R1),
   inorder(Right,R2),
   append(R1,[X|R2],R).

postorder(tree(X, L, R), Xs):-
  postorder(L, Ls),
  postorder(R, Rs),
  append(Ls, Rs, Xs1),
  append(Xs1, [X], Xs).
  postorder(_, []).
```

Output:-

9 ?- tree(M), inorder(M,L).
M = tree(5, tree(4, tree(2, nil, nil), tree(10, nil, nil)), tree(3, nil, nil)),
L = [2, 4, 10, 5, 3].

11 ?- tree(M), preorder(M,L).
M = tree(5, tree(4, tree(2, nil, nil), tree(10, nil, nil)), tree(3, nil, nil)),
L = [5, 4, 2, 10, 3] .

12 ?- tree(M), postorder(M,L).
M = tree(5, tree(4, tree(2, nil, nil), tree(10, nil, nil)), tree(3, nil, nil)),
L = [2, 10, 4, 3, 5]

Q.16 WAP to check whether a given element belongs to a binary tree.

Program:-

```
mem_btree(X,b_tree(X,L,R)).
mem_btree(X,b_tree(Y,L,R)):- mem_btree(X,L).
mem_btree(X,b_tree(Y,L,R)):-mem_btree(X,R).
```

Output:-

13 ?- [q16].
% q16 compiled 0.02 sec, 61 clauses
true.

14 ?-
mem_btree(X,b_tree(35,b_tree(25,b_tree(10,void,void),void),b_tree(62,void,void))).
X = 35 ;
X = 25 ;
X = 10 ;
X = 62 ;
false.

15 ?-
mem_btree(25,b_tree(35,b_tree(25,b_tree(10,void,void),void),b_tree(62,void,void))).
true .

16 ?-
mem_btree(65,b_tree(35,b_tree(25,b_tree(10,void,void),void),b_tree(62,void,void))).
false.

Q.17 WAP for Quick Sort using cut.

Program:-

```
gtq(X,Y) :- X @> Y.
quicksort( [],[] ).
quicksort( [X | Tail], Sorted) :-
split( X, Tail, Small, Big),quicksort( Small, SortedSmall),
quicksort( Big, SortedBig),conc1( SortedSmall, [X | SortedBig], Sorted).

split( _, [], [], []).

split( X,[Y | Tail], [Y | Small], Big) :-
      gtq( X, Y),!,
      split( X, Tail, Small, Big).

split( X, [Y | Tail], Small, [Y | Big] ) :-
      split( X, Tail, Small, Big).

conc1([],L,L).

conc1( [X | L1], L2, [X | L3]) :-
      conc1( L1, L2, L3).
```

Output:-

```
17 ?- [q17].
% q17 compiled 0.00 sec, 9 clauses
true.

18 ?- quicksort([1,2,23,10,6,21], Z).
Z = [1, 2, 6, 10, 21, 23] .

19 ?- quicksort([67,73,2,19,21,45], Z).
Z = [2, 19, 21, 45, 67, 73] .
```

Q.18 WAP to solve the following Monkey and Banana problem:

A hungry monkey finds himself in a room in which a bunch of bananas is hanging from the ceiling. The monkey, unfortunately, cannot reach the banana. However, in the room there are also a chair and a stick. The ceiling is just the right height so that a monkey standing on a chair could knock the bananas down with the stick. The monkey knows how to move around, carry other things around, reach for the bananas, and wave a stick in the air. What is the best sequence of actions for the monkey to take to acquire lunch?

Program:-

```
perform(grasp,
      state(middle, middle, onbox, hasnot),
      state(middle, middle, onbox, has)).
```

```
perform(climb,
        state(MP, BP, onfloor, H),
        state(MP, BP, onbox, H)).

perform(push(P1,P2),
        state(P1, P1, onfloor, H),
        state(P2, P2, onfloor, H)).

perform(walk(P1,P2),
        state(P1, BP, onfloor, H),
        state(P2, BP, onfloor, H)).
getfood(state(_,_,_,has)).

getfood(S1):- perform(Act, S1, S2),
              nl, write('In '),  write(S1),
              nl, write('    try '), write(Act),
              getfood(S2).
```

Output:-

```
5 ?- [q9].
% q9 compiled 0.00 sec, 7 clauses
true.

6 ?- getfood(state(atdoor,nearwindow,onfloor,hasnot)).

In state(atdoor,nearwindow,onfloor,hasnot)
  try climb
In state(atdoor,nearwindow,onfloor,hasnot)
  try walk(atdoor,_G780)
In state(_G780,nearwindow,onfloor,hasnot)
  try climb
In state(nearwindow,nearwindow,onfloor,hasnot)
  try push(nearwindow,_G788)
In state(_G788,_G788,onfloor,hasnot)
  try climb
In state(middle,middle,onbox,hasnot)
  try grasp
true
```

Q. 19 WAP to solve the 4-3 Gallon Water Jug Problem.

Water Jug Problem: We have a four gallon jug of water and a three gallon jug of water and there is a tap that can be used to fill the jugs with water. The challenge of the problem is to be able to put exactly two gallons of water in the four gallon jug, even though there are no markings on the jugs.

Program:-

```
solution(P) :-
   path(0, 0, [state(0, 0)], P).

path(2, 0, [state(2, 0)|_], []).

path(0, 2, C, ['Pour 2 gallons from 3-Gallon jug to 4-gallon.'|R] ) :-
not(member(state(2, 0), C)),
path(2, 0, [state(2, 0)|C], R).

path(X, Y, C, ['Fill the 4-Gallon Jug.'|R]) :-
X < 4,
not(member(state(4, Y), C)),
path(4, Y, [state(4, Y)|C], R).

path(X, Y, C, ['Fill the 3-Gallon Jug.'|R]) :-
Y < 3,
not(member(state(X, 3), C)),
path(X, 3, [state(X, 3)|C], R).

path(X, Y, C, ['Empty the 4-Gallon jug on ground.'|R]) :-
X > 0,
not(member(state(0, Y), C)),
path(0, Y, [state(0, Y)|C], R).

path(X, Y, C, ['Empty the 3-Gallon jug on ground.'|R]) :-
Y > 0,
not(member(state(X, 0), C)),
path(X, 0, [state(X, 0)|C], R).


path(X, Y, C, ['Pour water from 3-Gallon jug to 4-gallon until it is full.'|R]) :-
X + Y >= 4,
X < 4,
Y > 0,
NEW_Y is Y - (4 - X),
not(member(state(4, NEW_Y), C)),
path(4, NEW_Y, [state(4, NEW_Y)|C], R).

path(X, Y, C, ['Pour water from 4-Gallon jug to 3-gallon until it is full.'|R]) :-
X + Y >=3,
X > 0,
Y < 3,
NEW_X is X - (3 - Y),
not(member(state(NEW_X, 3), C)),
path(NEW_X, 3, [state(NEW_X, 3)|C], R).

path(X, Y, C, ['Pour all the water from 3-Gallon jug to 4-gallon.'|R]) :-
X + Y =< 4,
```

```
Y > 0,
NEW_X is X + Y,
not(member(state(NEW_X, 0), C)),
path(NEW_X, 0, [state(NEW_X, 0)|C], R).

path(X, Y, C, ['Pour all the water from 4-Gallon jug to 3-gallon.'|R]) :-
X + Y =< 3,
X > 0,
NEW_Y is  X + Y,
not(member(state(0, NEW_Y), C)),
path(0, NEW_Y, [state(0, NEW_Y)|C], R).
```

Output:-

```
1 ?- [q19].
% q19 compiled 0.00 sec, 1 clauses
true.

2 ?- solution(X).
X = ['Fill the 4-Gallon Jug.', 'Fill the 3-Gallon Jug.', 'Empty the 4-Gallon jug on ground.', 'Pour all the water from 3-Gallon jug to 4-gallon.', 'Fill the 3-Gallon Jug.', 'Pour water from 3-Gallon jug to 4-gallon until it is full.', 'Empty the 4-Gallon jug on ground.', 'Pour 2 gallons from 3-Gallon jug to 4-gallon.']
```

Q. 20 WAP to solve the following Missionaries and Cannibals problem:

In the missionaries and cannibals problem, three missionaries and three cannibals must cross a river using a boat which can carry at most two people, under the constraint that, for both banks, if there are missionaries present on the bank, they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries). The boat cannot cross the river by itself with no people on board. And, in some variations, one of the cannibals has only one arm and cannot row.

Program:-

```
% Main control block and printing

find :-
  path([3,3,left],[0,0,right],[[3,3,left]],_).
output([]) :- nl, nl.
output([[A,B,String]|T]) :-
output(T),
  write(B), write(' ~~ '), write(A), write(': '), write(String), nl.


% Base case

path([A,B,C],[A,B,C],_,MoveList):-
nl,nl,output(MoveList).


% Recursive call to solve the problem
```

```
path([A,B,C],[D,E,F],Traversed,Moves) :-
  move([A,B,C],[I,J,K],Out),
  legal([I,J,K]),  % Don't use this move unless it's safe.
  not(member([I,J,K],Traversed)),
  path([I,J,K],[D,E,F],[[I,J,K]|Traversed],[ [[I,J,K],[A,B,C],Out] | Moves ]).

% Move commands and descriptions of the move

move([A,B,left],[C,B,right],'One missionary crosses the river') :-
  A > 0, C is A - 1.
move([A,B,left],[C,B,right],'Two missionaries cross the river') :-
  A > 1, C is A - 2.
move([A,B,left],[C,D,right],'One missionary and One cannibal cross the river') :-
  A > 0, B > 0, C is A - 1, D is B - 1.
move([A,B,left],[A,D,right],'One cannibal crosses the river') :-
  B > 0, D is B - 1.
move([A,B,left],[A,D,right],'Two cannibals cross the river') :-
  B > 1, D is B - 2.
move([A,B,right],[C,B,left],'One missionary returns from the other side') :-
  A < 3, C is A + 1.
move([A,B,right],[C,B,left],'Two missionaries return from the other side') :-
  A < 2, C is A + 2.
move([A,B,right],[C,D,left],'One missionary and One cannibal return from the other
side') :-
  A < 3, B < 3, C is A + 1, D is B + 1.
move([A,B,right],[A,D,left],'One cannibal returns from the other side') :-
  B < 3, D is B + 1.
move([A,B,right],[A,D,left],'Two cannibals return from the other side') :-
  B < 2, D is B + 2.

% Legal move definition where B is missionaries and A is cannibals:

legal([B,A,_]) :-
  (A =< B ; B = 0),
  C is 3-A, D is 3-B,
  (C =< D; D = 0).
Output:-
1 ?- [q20].
% q20 compiled 0.00 sec, 1 clauses
true.

2 ?- find.

[3,3,left] ~~ [2,2,right]: One missionary and One cannibal cross the river
[2,2,right] ~~ [3,2,left]: One missionary returns from the other side
[3,2,left] ~~ [3,0,right]: Two cannibals cross the river
[3,0,right] ~~ [3,1,left]: One cannibal returns from the other side
[3,1,left] ~~ [1,1,right]: Two missionaries cross the river
[1,1,right] ~~ [2,2,left]: One missionary and One cannibal return from the other side
```

[2,2,left] ~~ [0,2,right]: Two missionaries cross the river
[0,2,right] ~~ [0,3,left]: One cannibal returns from the other side
[0,3,left] ~~ [0,1,right]: Two cannibals cross the river
[0,1,right] ~~ [1,1,left]: One missionary returns from the other side
[1,1,left] ~~ [0,0,right]: One missionary and One cannibal cross the river
true