LAB-1 | Assignment

Aim: Study of Facts, Objects, Predicates and Variables.

1. Write a prolog program for the given facts.

Code:

```
domains
        name,property = symbol
predicates
        colour(name,property).
        shape(name,property).
        size(name,property).
clauses
        colour(b1,red).
        colour(b2,blue).
        colour(b3,yellow).
        shape(b1,square).
        shape(b2,circle).
        shape(b3,square).
        size(b1,small).
        size(b2,small).
        size(b3,large).
```

What will be the outcome of each of the following queries?

1) What is the shape of b3?
   I/P : Goal: shape(b3,Shape)
   O/P : Shape = square
   1 Solution
2) Which component is having a large size and yellow colour?
   I/P : Goal: size(Component,large) and colour(Component,yellow)
   O/P : Component = b3
   1 Solution

---

2. Find the answer to the following questions from given facts.

Code:

```
domains
predicates
        likes(symbol,symbol)
clauses
        likes(mary,food).
        likes(mary,wine).
        likes(john,wine).
        likes(john,mary).
```

I/P : Goal: likes(mary,food)
O/P : Yes

I/P : Goal: likes(john,wine)
O/P : Yes

I/P : Goal: likes(john,food)
O/P : No

1) John likes anything that Mary likes.
   I/P : Goal: likes(john,Like) and likes(mary,Like)
   O/P : Like=wine
   1 Solution

2) John likes anyone who likes wine.
   I/P : Goal: likes(john,X) and likes(X,wine)
   O/P : X=mary
   1 Solution

---

3. Find the answer to the following questions from given facts.

Code:

```
domains
predicates
        has(symbol,symbol)
        fruit(symbol)
clauses
        has(jack,apples).
        has(ann,plums).
        has(dan,monkey).
        fruit(apples).
        fruit(plums).
```

1) What jack has?
   I/P : Goal: has(jack,X)
   O/P : X=apples
   1 Solution

2) Does Jack have something?
   I/P : Goal: has(jack,_)
   O/P : Yes

3) Who has apples and Who has plums?
   I/P : Goal: has(X,apples) or has(X,plums)
   O/P : X=jack
   X=ann
   2 Solutions

4) Does someone have apples and plums?
   I/P : Goal: has(X,apples) and has(X,plums)
   O/P : No

5) Has Dan fruits?
   I/P : has(dan,X),fruit(X)
   O/P : No Solution

LAB 2 | Artificial Intelligence

Aim: Study of RULES & UNIFICATION.

1. Write a prolog program for the given facts and rules and answer the given question.

Code:

```
domains
        patient, indication, disease=symbol
predicates
        symptom(patient, indication).
        hypothesis(patient, disease).
clauses
        symptom("Parva",fever).
        symptom("Parva",rash).
        symptom("Parva",headache).
        symptom("Parva",runny_nose).
        symptom("Vidhi",chills).
        symptom("Vidhi",fever).
        symptom("Vidhi",headache).
        symptom("Vivan",runny_nose).
        symptom("Vivan",rash).
        symptom("Vivan",flu).

        hypothesis(Patient,measles):-symptom(Patient,fever),
                                     symptom(Patient,cough),
                                     symptom(Patient,conjunctivitis),
                                     symptom(Patient,rash).

        hypothesis(Patient,german_measles):-symptom(Patient,fever),
                                            symptom(Patient,headache),
                                            symptom(Patient,runny_nose),
                                            symptom(Patient,rash).

        hypothesis(Patient,flu):-symptom(Patient,fever),
                                 symptom(Patient, headache),
                                 symptom(Patient,body_ache),
                                 symptom(Patient,chills).

        hypothesis(Patient,common_cold):-symptom(Patient,headache),
                                         symptom(Patient,sneezing),
                                         symptom(Patient,sore_throat),
```

```
                                        symptom(Patient,chills),
                                        symptom(Patient,runny_nose).

hypothesis(Patient,mumps):-symptom(Patient,fever),
                           symptom(Patient,swollen_glands).

hypothesis(Patient,chicken_pox):-symptom(Patient,fever),
                                 symptom(Patient,rash),
                                 symptom(Patient,body_ache),
                                 symptom(Patient,chills).
```

Question : Identify patients with any particular disease based on rules and facts given above.

```
Goal: hypothesis(X,measles)
No Solution
Goal: symptom(X,fever),symptom(X,rash),symptom(X,headache),symptom(X,runny_nose)
X=Parva
1 Solution
Goal: hypothesis(X,chicken_pox)
No Solution
Goal: hypothesis("Vivan",flu)
No
Goal: symptom(X,headache)
X=Parva
X=Vidhi
2 Solutions
```

2. Write a program for a family tree given question which contains three predicates: male, female, parent.
   Make rules for family relations : father , mother, grandfather, grandmother, brother, sister, uncle, aunt, nephew and niece.

Code:

```
predicates
        male(symbol).
        female(symbol).
        parent(symbol,symbol).
        father(symbol,symbol).
        mother(symbol,symbol).
        wife(symbol,symbol).
        grandfather(symbol,symbol).
        grandmother(symbol,symbol).
        brother(symbol,symbol).
        sister(symbol,symbol).
        uncle(symbol,symbol).
        aunt(symbol,symbol).
        nephew(symbol,symbol).
        niece(symbol,symbol).

clauses
        male("Pandu").
        male("Nakula").
        male("Sahadeva").
        male("Arjuna").
        male("Bhima").
        male("Yudhishthira").
        male("Satanika").
        male("Shrutasena").
        male("Shrutakarma").
        male("Abhimanyu").
        male("Iravan").
        male("Babruvahana").
        male("Sutasoma").
        male("Prativindhya").

        female("Madri").
        female("Kunti").
        female("Draupadi").
        female("Subhadra").
        female("Ulupi").
        female("Chitrangada").
```

```
parent("Pandu","Nakula").
parent("Pandu","Sahadeva").
parent("Pandu","Arjuna").
parent("Pandu","Bhima").
parent("Pandu","Yudhishthira").
parent("Madri","Nakula").
parent("Madri","Sahadeva").
parent("Kunti","Arjuna").
parent("Kunti","Bhima").
parent("Kunti","Yudhishthira").
parent("Nakula","Satanika").
parent("Draupadi","Satanika").
parent("Sahadeva","Shrutasena").
parent("Draupadi","Shrutasena").
parent("Arjuna","Shrutakarma").
parent("Arjuna","Abhimanyu").
parent("Arjuna","Iravan").
parent("Arjuna","Babruvahana").
parent("Draupadi","Shrutakarma").
parent("Subhadra","Abhimanyu").
parent("Ulupi","Iravan").
parent("Chitrangada","Babruvahana").
parent("Bhima","Sutasoma").
parent("Draupadi","Sutasoma").
parent("Yudhishthira","Prativindhya").
parent("Draupadi","Prativindhya").

father(X,Y):-parent(X,Y),male(X).
mother(X,Y):-parent(X,Y),female(X).
wife(X,Y):-parent(X,Z),parent(Y,Z),
            male(X),female(Y).
grandfather(X,Y):-father(X,Z),father(Z,Y).
grandmother(X,Y):-mother(X,Z),father(Z,Y).
brother(X,Y):-father(A,X),father(A,Y),
                mother(B,X),mother(B,Y),
                male(X),not(X=Y).
sister(X,Y):-father(A,X),father(A,Y),
              mother(B,X),mother(B,Y),
              female(X),not(X=Y).
uncle(X,Y):-father(Z,Y),brother(X,Z).
aunt(X,Y):-father(Z,Y),brother(B,Z),wife(B,X).
nephew(X,Y):-father(Z,Y),brother(X,Z),
                male(X),male(Y).
niece(X,Y):-father(Z,Y),brother(X,Z),
               male(X),female(Y).
```

Output :

```
Goal: father("Pandu",Y)
Y=Nakula
Y=Sahadeva
Y=Arjuna
Y=Bhima
Y=Yudhishthira
5 Solutions
Goal: mother("Kunti",X)
X=Arjuna
X=Bhima
X=Yudhishthira
3 Solutions
Goal: grandfather(X,"Prativindhya")
X=Pandu
1 Solution
Goal: brother(X,"Arjuna")
X=Bhima
X=Yudhishthira
2 Solutions
Goal: uncle("Arjuna",Y)
Y=Sutasoma
Y=Prativindhya
2 Solutions
Goal: nephew("Bhima",Y)
Y=Shrutakarma
Y=Abhimanyu
Y=Iravan
Y=Babruvahana
Y=Prativindhya
5 Solutions
```

3. Write a prolog program for the given facts and rules, trace the given goals.

Code:

```
domains
        course, level, material, component, person = symbol
predicates
        is(course,level).
        available(course,material).
        has(course,component).
        takes(person,course).
        hypothesis(person,course).
clauses
        is("hardware","easy").
        is("logic","not easy").
        is("graphics","easy").

        has("graphics","8 credits").
        has("graphics","lab component").

        available("hardware","Books").
        available("database","Books").

        takes("Mary","compilers").

        hypothesis(X,Y):-takes(X,Y),is(Y,"easy"),available(Y,"Books").
        hypothesis(X,Y):-takes(X,Y),has(Y,"8 credits"),has(Y,"lab component").
```

Goals:

1. Does Mary take a graphics course?
   I/p & O/p:
   ```
   Goal: takes("Mary","graphics")
   No
   ```

2. Which course Mary takes?
   I/p & O/p:
   ```
   Goal: takes("Mary",X)
   X=compilers
   1 Solution
   ```

3. Who takes graphics course?

   <u>I/p & O/p:</u>

```
Goal: takes(X,"graphics")
No Solution
```

LAB 3 | Artificial Intelligence

Aim : To learn simple input and output predicates in prolog and to build rule based consultation program.

1. Predict the user‟s nature based on colour user likes.

Code:

```
domains
        User, name = string
        colour_name, characteristic_name = symbol

predicates
        colour(colour_name).
        likes(name).
        response(char).
        characteristic(colour_name,characteristic_name).
        result(colour_name,characteristic_name).
        go.

clauses

        colour(red).
        colour(orange).
        colour(yellow).
        colour(green).
        colour(blue).
        colour(purple).
        colour(brown).
        colour(grey).
        colour(black).

        characteristic(red,very_social).
        characteristic(red,assertive).
        characteristic(red,energetic).
        characteristic(red,moody).
        characteristic(red,impulsive).
        characteristic(red,sympathetic).
        characteristic(red,easiy_swayed).
        characteristic(red,optimist).
        characteristic(red,complainer).
        characteristic(red,brave).
```

```
characteristic(orange,good_natured).
characteristic(orange,very_social).
characteristic(orange,easiy_swayed).
characteristic(orange,loyal).
characteristic(orange,pure_heart).
characteristic(orange,good_work_ethics).

characteristic(yellow,very_imaginative).
characteristic(yellow,urge_to_help).
characteristic(yellow,free_spirit).
characteristic(yellow,shy).
characteristic(yellow,wise).
characteristic(yellow,mental_loner).
characteristic(yellow,keep_secrets_of_friends).

characteristic(green,sensitive).
characteristic(green,good_citizen).
characteristic(green,etiquette).
characteristic(green,frank).
characteristic(green,moral).
characteristic(green,reputable).
characteristic(green,deep_affection_towards_your_family).

characteristic(blue,deliberate).
characteristic(blue,introspective).
characteristic(blue,sensitive).
characteristic(blue,loyal).
characteristic(blue,sober).
characteristic(blue,dreamer).
characteristic(blue,ego).

characteristic(purple,good_mind).
characteristic(purple,observer).
characteristic(purple,angry).
characteristic(purple,creative).
characteristic(purple,appreciator).

characteristic(brown,good_citizen).
characteristic(brown,clever).
characteristic(brown,stubborn).
characteristic(brown,dependable).
characteristic(brown,not_implusive).
characteristic(brown,bargainer).

characteristic(grey,cautious).
characteristic(grey,compromiser).
characteristic(grey,peaceful).
```

```
characteristic(black,above_average).
characteristic(black,conventional).
characteristic(black,decent).
characteristic(black,polite).
characteristic(black,regal).


result(X,Y):-characteristic(X,Y),
            write("\t=",Y,"\n"),
            fail.

likes(User):-colour(X),
            write("Does ",User," like ",X," colour (y/n) ? : "),
            response(Flag),
            Flag='y',!,
            write(User," is :\n"),
            result(X,_),
            write("\n").

go:-write("Enter Your Name : "),nl,
   readln(User),
   likes(User).

response(Flag) :-readchar(Flag),
write(Flag),nl.
```

Output:

```
Goal: go
Enter Your Name :
Raj
Does Raj like red colour (y/n) ? : n
Does Raj like orange colour (y/n) ? : n
Does Raj like yellow colour (y/n) ? : n
Does Raj like green colour (y/n) ? : n
Does Raj like blue colour (y/n) ? : y
Raj is :
        =deliberate
        =introspective
        =sensitive
        =loyal
        =sober
        =dreamer
        =ego
No
```

2. Predict user‟s health based on habits user practices.

Code :

```
domains
        User,name=string
        status,habit = symbol

predicates
        health(string,status).
        habitof(name,habit).
        response(char).
        go.

clauses
        go:-write("Enter Your Name : "),nl,
           readln(User),
           health(User,Status),
           write(User,"'s health is ",Status),nl.

        go:-write("Sorry, I can't say about your health."),nl.

        habitof(User,"regular smoking"):-
                                write("Does ",User," have habit of regular smoking ?
(y/n)"),
                                response(Reply),
                                Reply='y'.

        habitof(User,"excessive drinking regularly"):-
                                write("Does ",User," have habit of excessive drinking
regularly ? (y/n)"),
                                response(Reply),
                                Reply='y'.

        habitof(User,"taking drugs"):-
                                write("Does ",User," have habit of taking drugs ? (y/n)"),
                                response(Reply),
                                Reply='y'.

        habitof(User,"eating oily food"):-
                                write("Does ",User," have habit of eating oily food ? (y/n)"),
                                response(Reply),
                                Reply='y'.
```

```
        habitof(User,"taking too much sugar with foods"):-
                                        write("Does ",User," have habit of taking too much sugar
with foods ? (y/n)"),
                                        response(Reply),
                                        Reply='y'.

        habitof(User,"sleep hours are less"):-
                                        write("Does ",User," acts like an ownl ? (y/n)"),
                                        response(Reply),
                                        Reply='y'.

        habitof(User,"drinking milk regularly"):-
                                        write("Does ",User," have habit of drinking milk regularly ?
(y/n)"),
                                        response(Reply),
                                        Reply='y'.

        habitof(User,"eating green vegetables or eggs in meal"):-
                                        write("Does ",User," have habit of eating green
vegetables or eggs in meal ? (y/n)"),
                                        response(Reply),
                                        Reply='y'.

        habitof(User,"regular exercise"):-
                                        write("Does ",User," have habit of regular exercise ?
(y/n)"),
                                        response(Reply),
                                        Reply='y'.

        habitof(User,"drinking enough water during day"):-
                                        write("Does ",User," have habit of drinking enough water
during day ? (y/n)"),
                                        response(Reply),
                                        Reply='y'.

        habitof(User,"regular sufficient sleep hours"):-
                                        write("Does ",User," have habit of regular sufficient sleep
hours ? (y/n)"),
                                        response(Reply),
                                        Reply='y'.

        habitof(User,"regular walk"):-
                                        write("Does ",User," have habit of regular walk ? (y/n)"),
                                        response(Reply),
                                        Reply='y'.

        habitof(User,"brushing teeth and washing hair and using showers regularly"):-
```

```
                                        write("Does ",User," have habit of brushing teeth and
washing hair and using showers regularly ? (y/n)"),
                                        response(Reply),
                                        Reply='y'.

        health(User,bad):-habitof(User,"regular smoking"),
                          habitof(User,"excessive drinking regularly"),
                          habitof(User,"taking drugs"),
                          habitof(User,"eating oily food"),
                          habitof(User,"taking too much sugar with foods"),
                          habitof(User,"sleep hours are less").

        health(User,good):-habitof(User,"drinking milk regularly"),
                           habitof(User,"eating green vegetables or eggs in meal"),
                           habitof(User,"drinking enough water during day"),
                           habitof(User,"regular exercise"),
                           habitof(User,"regular sufficient sleep hours"),
                           habitof(User,"regular walk"),
                           habitof(User,"brushing teeth and washing hair and using showers
regularly").

        health(User,moderate):-habitof(User,"eating oily food"),
                               habitof(User,"regular walk"),
                               habitof(User,"taking too much sugar with foods").


   response(Reply):-readchar(Reply),
                    write(Reply),nl.
```

Output :

```
Enter Your Name :
Raj
Does Raj have habit of regular smoking ? (y/n)y
Does Raj have habit of excessive drinking regularly     ? (y/n)y
Does Raj have habit of taking drugs ? (y/n)n
Does Raj have habit of drinking milk regularly ? (y/n)y
Does Raj have habit of eating green vegetables or eggs in meal ? (y/n)y
Does Raj have habit of drinking enough water during day ? (y/n)n
Does Raj have habit of eating oily food ? (y/n)y
Does Raj have habit of regular walk ? (y/n)y
Does Raj have habit of taking too much sugra with foods ? (y/n)y
Raj's health is moderate
Yes
```

LAB 4 | Artificial Intelligence

Aim : To learn arithmetic operations and recursion in Prolog.

1. Write a prolog program to find roots (real roots only) of quadratic equations.

Code:

```
predicates
        go.
        find_roots(real,real,real).

clauses
        go:-
                write("Enter a,b,c : "),nl,
                readreal(A),
                readreal(B),
                 readreal(C),
                 D = (B*B) - (4*A*C),
                 find_roots(D,A,B).

        find_roots(D,A,B):-
                        D=0,
                        X=(-B)/(2*A),
                        write("X = "),write(X),nl;

                        D>0,
                        X1=(-B + sqrt(D))/(2*A),
                        X2=(-B - sqrt(D))/(2*A),
                        write("x1 = " ),write(X1),
                        write(" x2 = "),write(X2),nl;

                        D<0,
                        write("Not possible to find roots."),nl.
```

Output:

Case I:

```
Goal : go
Enter a,b,c :
1
3
2
x1 = -1 x2 = -2
Yes
```

Case II:

```
Goal : go
Enter a,b,c:
1
0
1
Not possible to find roots.
Yes
```

2. Write a prolog program to implement a logon routine. This routine must ask username and password and verify with a pair of username and password available (i.e. stored as clauses) as facts. On a successful match system display “welcome message” and on an unsuccessful attempt the user is allowed 3 times to re enter valid credentials. If a user enters incorrect credentials continuously 3 times then the system exits with “unsuccessful attempt message”.

Code:

```
domains
        person, password = symbol

predicates
        login.
        count(integer).
        logon(person,password).
        message(integer).

clauses
        logon("Raj","raj5126").
        logon("Parth","parth5820").
        login:-
                write("Welcome ! Please Login"),nl,
                count(3),
                write("Login Unsuccessful"),nl.

        login:-write("Login Successful"),nl.

        count(X):-
                X<>0,
                write("Enter user name : "),nl,
                readln(User),
                write("Enter password : "),nl,
                readln(Passwd),
                not(logon(User,Passwd)),
                Y=X-1,
                message(Y),
                count(Y).
        count(0).

        message(X):-X<>0,write("Login Failed, Please Try Again Later"),nl.

        message(0).
```

Output:

Case I:

```
Goal : login
Welcome ! Please Login
Enter user name :
Raj
Enter password :
raj5126
Login Successful
Yes
```

Case II:

```
Goal : login
Welcome ! Please Login
Enter user name:
Dhruv
Enter password :
dp0105
Login Failed, Please Try Again Later
Enter user name:
Dhruv
Enter password :
dp2002
Login Failed, Please Try Again Later
Enter user name:
Parth
Enter password :
parth5820
Login Unsuccessful
Yes
```

3. Write a prolog program to find the factorial of a given number.

Code:

```
predicates
        factorial(integer,integer).
clauses
        factorial(0,1).
        factorial(N,F):-
                N>0,
                N1=N-1
                factorial(N1,F1),
                F=N*F1.
```

Output:

```
Goal : factorial(5,F)
F=120
1 Solution
Goal : factorial(0,F)
F=1
1 Solution
```

4. Write a prolog program to find the sum of first n numbers.

Code:

```
predicates
        sum(integer,integer).
clauses
        sum(0,0).
        sum(N,Sum):-
                N>0,
                N1=N-1,
                sum(N1,R1),
                Sum=R1+N.
```

Output:

```
Goal : sum(10,X)
X=55
1 Solution
```

5. Write a prolog program to print the nth term of Fibonacci series.

Code:

```
predicates
        fib(integer,integer).

clauses
        fib(0,0):-!.
        fib(1,1):-!.
        fib(N,Result):-
                        N1=N-1,
                        N2=N-2,
                        fib(N1,Result1),
                        fib(N2,Result2),
                        Result=Result1+Result2.
```

Output:

```
Goal : fib(2,X)
X=1
1 Solution
Goal : fib(7,X)
X=13
1 Solution
```

6. Write a prolog program to print Fibonacci series up-to nth term.

Code:

```
predicates
        fib(integer,integer,integer).
        go.

clauses
        go:-
                write("Enter number : "),
                readreal(N),
                A=0,
                B=1,
                write(A),write(' '),write(B),write(' '),
                fib(N,A,B).

        fib(N,A,B):-
                N<2,nl;
                C=A+B,
                write(C),
                write(" "),
                D=B,
                E=C,
                N1=N-1,
                fib(N1,D,E).
```

Output:

```
Goal : go
Enter number : 5
0 1 1 2 3 5
Yes
Goal : go
Enter number : 1
0 1
Yes
```

LAB 5 | Artificial Intelligence

Aim : To study about controlling execution in prolog using cut and fail predicate

Fail:

- Failure can be forced in any rule by using the built-in fail predicate.
- The fail forces backtracking in an attempt to unify with other clauses.

Cut:

- The primary purpose of cut is to prevent or block backtracking based on a specified condition.
- The cut predicate is specified as an exclamation point(!).

1. Implement a prolog program to find minimum and maximum of two integers using cut and/ or fail predicate. Program must have three arguments and it must handle all cases.

Code :

```
predicates
        max(integer,integer,integer).
        min(integer,integer,integer).
clauses
        max(X, Y, Max):-X>=Y,!,Max=X;Max=Y.
        min(X, Y, Min):-X<=Y,!,Min=X;Min=Y.
```

Output:

```
Goal : max(100,200,Max)
Max=200
1 Solution

Goal : min(100,200,Min)
Min=100
1 Solution

Goal : max(100,200,100)
No

Goal : min(100,200,100)
Yes
```

2. Write a prolog program to verify that a given year is leap year or not using cut and/ or fail predicate.
   Note: A year is a leap year if it is divisible by 4, but century years are not leap years unless they are divisible by 400. So, the years 1700, 1800, and 1900 were not leap years, but the year 2000 was.

Code:

```
domains
        year=integer

predicates
        leap_check(year).
        check(year).

clauses
        leap_check(Year):-
                Year mod 4 = 0,
                Year mod 100 = 0,
                Year mod 400 = 0.

        leap_check(Year):-
                Year mod 4 = 0,
                Year mod 100<>0.

        check(Year):-
                Year<0,!,
                write("Year cannot be negative"),nl.

        check(Year):-
                leap_check(Year),!,
                write(Year, " is a leap year."),nl;write(Year, " is not a leap year.").
```

Output:

```
Goal : check(1700)
1700 is not a leap year.
Yes

Goal : check(2000)
2000 is  a leap year.
Yes
```

3. Write a prolog program to verify that a given number is prime or not using cut and/ or fail predicate.

Code:

```
predicates
        isprime(integer)
        check(integer,integer)

clauses
        isprime(0):-!,fail.
        isprime(1):-!,fail.
        isprime(2):-!.
        isprime(X):-check(X,2).

        check(X,Y):-Y>sqrt(X),!.
        check(X,Y):-Y<=sqrt(X),X mod Y=0,!,fail.
        check(X,Y):-YY=Y+1,check(X,YY).
```

Output :

```
Goal : isprime(3)
Yes

Goal : isprime(25)
No
```

LAB 6 | Artificial Intelligence

Aim : Write a prolog program to check whether a number is a member of a given list or not.

1. Write a prolog program to check whether a number is a member of a given list or not.

Code :

```
domains
        list = integer*

predicates
        member(integer,list)

clauses
        member(X,[X|_]).
        member(X,[_|T]):-member(X,T).
```

Output:

```
Goal : member(1, [ 1, 2, 3 ] )
Yes

Goal : member(0, [ 1, 2, 3 ] )
No
```

2. Write a prolog program to concatenate two lists giving a third list.

Code :

```
domains
        list = integer*

predicates
        append(list,list,list).

clauses
        append([],L,L).
        append([X|L1],L2,[X|L3]):-append(L1,L2,L3).
```

Output:

```
Goal : append( [ 1, 2 ], [ 3, 4 ] ,X )
X=[ 1, 2, 3, 4 ]
1 Solution

Goal : append( [ 1, 2 ], [ 3, 4 ], [ 1, 2, 3, 4 ] )
Yes
```

3. Write a prolog program to find the last element in a given list.

Code:

```
domains
        list=integer*

predicates
        last_element(list,integer)

clauses
        last_element([Z],X):-Z=X.
        last_element([_|T],X):-last_element(T,X).
```

Output:

```
Goal : last_element( [ 1, 2, 3, 4 ], X )
X=4
1 Solution

Goal : last_element( [ 1, 2, 3 ], 3 )
Yes
```

4. Write a prolog program to reverse a list.

Code:

```
domains
        list=integer*

predicates
        reverse(list,list,list).

clauses
        reverse([],InputList,InputList).
        reverse([H|T],List1,List2):-reverse(T,[H|List1],List2).
```

Output:

```
Goal : reverse( [ 1, 2, 3 ], [ ], Z)
Z=[ 3, 2, 1 ]
1 Solution

Goal : reverse( [ 1, 2, 3 ], [ ], [ 2, 3 ] )
No
```

5. Write a prolog program to find the nth element of a list.

Code:

```
domains
        list=integer*

predicates
        nth_element(list,integer)

clauses
        nth_element([H|_],1):-write(H),nl.
        nth_element([_|T],N):-NN=N-1,
                                nth_element(T,NN).
```

Output:

```
Goal : nth_element( [ 10, 20, 30, 40, 50 ], 3 )
30
Yes
```

6. Write a prolog program to split a list in two lists such that one list contains negative numbers and one contains positive numbers.

Code:

```
domains
        list=integer*

predicates
        split_list(list,list,list).

clauses
        split_list([],[],[]).
        split_list([X|L],[X|L1],L2):-X>=0,!,split_list(L,L1,L2).
        split_list([X|L],L1,[X|L2]):-split_list(L,L1,L2).
```

Output:

```
Goal : split_list( [ 1, -1, -2, 2 ], P, N )
P=[ 1, 2 ], N=[ -1, -2 ]

Goal : split_list( [ 1, -1, -2, 2 ], [ 1, -2 ], [ 2, -1 ] )
No
```

LAB 7 | Artificial Intelligence

Aim : Study Compound objects and Functors in PROLOG.

1. Modify the sample program II so that it will also print the birth dates of the people listed. Next, add telephone numbers to the report.

Code :

```
domains
        name = person(symbol,symbol)
        birthday = b_date(symbol,integer,integer)
        ph_num = symbol

predicates
        phone_list(name,symbol,birthday)
        get_months_birthdays
        convert_month(symbol,integer)
        check_birthday_month(integer,birthday)
        write_person_birthdate_mobileno(name,birthday,ph_num)

clauses
        get_months_birthdays:-
                        write("******************** This Month's Birthday List ********************"),nl,
                        write("First name\tLast Name\tBirth Date\tMobile No\n"),
                        write("*********************************************************************"),nl,
                        date(_, This_month, _),
                        phone_list(Person, Mobile, Date),
                        check_birthday_month(This_month, Date),
                        write_person_birthdate_mobileno(Person,Date,Mobile),
                        fail.

        get_months_birthdays:-
                                write("\n\n Press any key to continue: "),nl,
                                readchar(_).


write_person_birthdate_mobileno(person(First_name,Last_name),b_date(M,D,Y),Mobile):-
                        write(" ",First_name,"\t\t ",Last_name,"\t\t",M,"-",D,"-",Y,"\t",Mobile),nl.

check_birthday_month(Mon,b_date(Month,_,_)):-
                                convert_month(Month,Month1),
                                Mon = Month1.
```

```
phone_list(person(apurva, mehta), "767-8463", b_date(jan, 13, 1955)).
phone_list(person(apurva, shah), "438-8400", b_date(feb, 04, 1985)).
phone_list(person(apurva, parikh), "555-5653", b_date(mar, 22, 1935)).
phone_list(person(apurva, doshi), "767-2223", b_date(apr, 04, 1951)).
phone_list(person(apurva, joshi), "555-1212", b_date(may, 31, 1962)).
phone_list(person(apurva, baxi), "438-8400", b_date(jun, 13, 1980)).
phone_list(person(apurva, dave), "767-8463", b_date(jun, 22, 1986)).
phone_list(person(apurva, bhatt), "555-5653", b_date(jul, 22, 1981)).
phone_list(person(apurva, patel), "767-2223", b_date(aug, 13, 1981)).
phone_list(person(apurva, dangar), "438-8400", b_date(sep, 22, 1981)).
phone_list(person(apurva, pandya), "438-8400", b_date(sep, 31, 1952)).
phone_list(person(apurva, vaishnav), "555-1212", b_date(nov, 22, 1984)).
phone_list(person(apurva, gor), "767-2223", b_date(sep, 04, 1987)).
phone_list(person(apurva, kanani), "438-8400", b_date(dec, 31, 1981)).

convert_month(jan, 1).
convert_month(feb, 2).
convert_month(mar, 3).
convert_month(apr, 4).
convert_month(may, 5).
convert_month(jun, 6).
convert_month(jul, 7).
convert_month(aug, 8).
convert_month(sep, 9).
convert_month(oct, 10).
convert_month(nov, 11).
convert_month(dec, 12).
```

Output:

```
Goal : get_months_birthdays
************************************This Month's Birthday List************************************
First name        Last Name          Birth Date        Mobile No
*****************************************************************************************************
apurva            danagar            sep-22-1981       438-8400
apurva            pandya             sep-31-1952       438-8400
apurva            gor                sep-4-1987        767-2223
```

2. Write a prolog program for an IT company that stores employee details like Name, Address, Department,Position, Salary. Use compound objects to properly formulate the representation of each employee's details. Find out,

   I. employee(s) with salary higher than a threshold.
   II. employee(s) available in a particular department.
   III. employee(s) holding a particular position.

Code :

```
domains
        name = person(first,last)
        location = address(street, city, state, zip)
        first, last, street, city, state, zip, department, position = symbol
        salary =integer

predicates
        employee(name,location,department,position,salary).
        employee_with_salary_higher_than_5000.
        employee_available_particular_department(department).
        employee_with_particular_position(position).
        write_name_salary(name,salary).
        write_name(name).

clauses

employee(person("Raj","Panchal"),address("HONEY
PARK","Surat","Gujarat","395009"),"Development","Senior Head",15000).

employee(person("Parth","Patel"),address("Dungari","Valsad","Gujarat","324856"),"Developmen
t","Senior Head",12000).

employee(person("Siddhi","Shah"),address("Ring
Road","Vadodra","Gujarat","375002"),"Marketing","Junior",700).

employee(person("Rutu","Joshi"),address("M G Road","Bhavnagar","Gujarat","362009"),"Human
Resource","Fresher",20000).

employee(person("Pranav","Patel"),address("Jalapor","Navsari","Gujarat","322009"),"Quality
Assurance","Senior Head",4000).

employee(person("Prachi","Shah"),address("Kabilpor","Navsari","Gujarat","322059"),"Quality
Assurance","Junior",2000).
```

```
employee_with_salary_higher_than_5000:-
                write("Employee with 50000 salary"),nl,
                write("--------------------------"),nl,
                employee(Name,_,_,_,Salary),
                Salary>5000,
                write_name_salary(Name,Salary),
                fail.

employee_available_particular_department(Department):-
        write("Employee with ",Department," Department"),nl,
        write("--------------------------"),nl,
        employee(Name,_,Department,_,_),
        write_name(Name),
        fail.

employee_with_particular_position(Position):-
                write("Employee with ",Position," Position"),nl,
                write("--------------------------"),nl,
                employee(Name,_,_,Position,_),
                write_name(Name),
                fail.

write_name_salary(person(First, Last), Salary):-
                write(First," ",Last,"\t",Salary),nl.

write_name(person(First,Last)):-
        write(First," ",Last),nl.
```

Output :

```
Goal : employee_with_salary_higher_than_5000
Employee with 5000 salary
----------------------------------------
Raj Panchal      15000
Parth Patel      12000
Rutu Joshi       20000
No

Goal : employee_available_particular_department("Development")
Employee with Development Department
------------------------------------------------------------
Raj Panchal
Parth Patel
No
```

```
Goal : employee_with_particular_position("Junior")
Employee with Junior Position
----------------------------------------
Siddhi Shah
Prachi Shah
No
```

3. Try the following link and verify whether the system is intelligent or not and justify your answer.
   www.manifestation.com/neurotoys/eliza.php3

- ELIZA emulates a Rogerian psychotherapist.
- ELIZA has almost no intelligence whatsoever, only tricks like string substitution and canned responses based on keywords.
- The illusion of intelligence works best, however if you limit your conversation to talking about yourself and your life.
- Hence , ELIZA is a dump.

LAB 8 | Artificial Intelligence

Aim : Database Handling in Prolog.

1. Write a prolog program to create applications like “marriage bureau” using dynamic databases and compound objects, and use files to store data.

Code :

```
domains
    name,gender,address=symbol
    phone=string
    age=integer

database
    person(name,age,gender,address,phone)

predicates
    writePerson.
    searchByName(name).
    searchByPhone(phone).
    search.
    openDB.
    deleteByName(name).
    deleteByPhone(phone).
    updateByName(name).
    updateByPhone(phone).

clauses
    openDB:-consult("d:\database.txt").

    writePerson:-
        readln(Name),readint(Age),readln(Gender),readln(Address),readln(Phone),
        asserta(person(Name,Age,Gender,Address,Phone)),save("d:\database.txt").

    search:-
                retract(person(Name,Age,Gender,Address,Phone)),
                write(Name),nl, write(Age),nl,write(Gender),nl,write(Address),nl,write(Phone),nl,fail.

    searchByName(Name1):-
            retract(person(Name1,Age,Gender,Address,Phone)),
            write(Name1),nl, write(Age),nl,write(Gender),nl,write(Address),nl,write(Phone),nl,fail.

    searchByPhone(Phone1):-
```

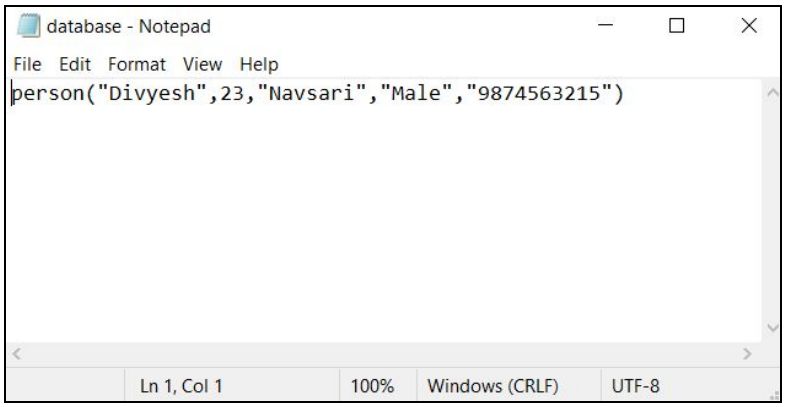
```
        retract(person(Name,Age,Gender,Address,Phone1)),
        write(Name),nl, write(Age),nl,write(Gender),nl,write(Address),nl,write(Phone1),nl,fail.

deleteByName(Name1):-
      retract(person(Name1,_,_,_,_)),
      save("database.txt"),nl.

 deleteByPhone(Phone1):-
     retract(person(_,_,_,_,Phone1)),
     save("database.txt"),nl.

  updateByName(X):-
    retract(person(X,_,_,_,_)),readint(Age),readln(Gender),readln(Address),readln(Phone),
    asserta(person(X,Age,Gender,Address,Phone)),save("d:\database.txt").

  updateByPhone(X):-
      retract(person(_,_,_,_,X)),readln(Name),readint(Age),readln(Gender),readln(Address),
      asserta(person(Name,Age,Gender,Address,X)),save("d:\database.txt").
```

Output :

```
Goal : openDB
Yes

Goal : writePerson
Raj
21
Surat
Male
7874716190

Goal : search
Raj
21
Surat
Male
7874716190

Goal : searchByName(“Parth”)
Parth
20
Male
Valsad
9265928833
No
```

```
Goal : searchByPhone("7874716190")
Raj
21
Male
Surat
7874716190
No

Goal : updateByName("Divyesh")
23
Navsari
Male
9874563215

Goal : searchByName("Divyesh")
Divyesh
23
Navsari
Male
9874563215
No
```

Database File :



2. Code for ProLog Program of searching a students data when Name or a phone no is input in Artificial Intelligence.

Code:

```
domains
        name,address = symbol
        phone = string
        l = integer*

predicates
        start
        repeat
        selectItem(integer)
        studentData
        subjectL(l)
        searchByName(name)
        searchByPhone(phone)

database
        studentDB(name,address,phone,l)


goal
        clearwindow,
        makewindow(1,7,7,"Search Student Detail",0,0,25,80),
        start.

clauses
        repeat.
                repeat:-
        repeat.

        start:-
                repeat,
                write("\n0.Exit"),
                write("\n1.Enter student data"),
                write("\n2.Search by Name"),
                write("\n3.Search by Phone number"),
                write("\n4.Show all Student Data"),
                write("\nEnter your choice::"),
                readint(Choice),
                selectItem(Choice),
                Choice=0.
```

```
selectItem(0).

selectItem(1):-
        studentData,
        fail.

selectItem(2):-
        write("\nEnter your name::"),
        readln(Name),
        searchByName(Name),
        fail.

selectItem(3):-
        write("\nEnter the phone no::"),
        readln(Phone),
        searchByPhone(Phone),
        fail.

selectItem(4):-
        studentDB(Name,Address,Phone,Marks),
        write(Name," ",Address," ",Phone," ",Marks),nl,
        fail.

studentData:-
        write("\nEnter the name of the student::"),
        readln(Name),
        write("\nEnter the address of the student::"),
        readln(Address),
        write("\nEnter the phone number of the student::"),
        readln(Phone),
        write("\nEnter the five subject marks of the student"),
        subjectL(Marks),
        assert(studentDB(Name,Address,Phone,Marks)).

subjectL(Marks):-
        write("\nC ::"),
        readint(C),
        write("\nC++ ::"),
        readint(CC),
        write("\nVB ::"),
        readint(VB),
        write("\nJAVA ::"),
        readint(Java),
        write("\nPROLOG ::"),
        readint(Prolog),
        Marks=[C,CC,VB,Java,Prolog].
```

```
searchByName(Name1):-
        studentDB(Name1,Address,Phone,Marks),
        write("\nName::",Name1),
        write("\nAddress::",Address),
        write("\nPhone::",Phone),
        write("\nMarks[C,C++,VB,Java,Prolog]::",Marks).

searchByPhone(Phone1):-
        studentDB(Name,Address,Phone1,Marks),
        write("\nName::",Name),
        write("\nAddress::",Address),
        write("\nPhone::",Phone1),
        write("\nMarks[C,C++,VB,Java,Prolog]::",Marks).
```

Output:

```
                    Search Student Detail

0.Exit
1.Enter student data
2.Search by Name
3.Search by Phone number
4.Show all Student Data
Enter your choice::1

Enter the name of the student::Raj

Enter the address of the student::Surat

Enter the phone number of the student::7874716190

Enter the five subject marks of the student
C :: 45

C++ ::52

VB :: 50

JAVA :: 65

PROLOG :: 70

0.Exit
1.Enter student data
2.Search by Name
```

```
3.Search by Phone number
4.Show all Student Data
Enter your choice::2

Enter your name :: Raj

Name :: Raj
Address :: Surat
Phone :: 7874716190
Marks[C,C++,VB,Java,Prolog] :: [45,52,50,65,70]

0.Exit
1.Enter student data
2.Search by Name
3.Search by Phone number
4.Show all Student Data
Enter your choice::4

Raj Surat 7874716190 [45,52,50,65,70]

0.Exit
1.Enter student data
2.Search by Name
3.Search by Phone number
4.Show all Student Data
Enter your choice::0

Press the SPACEbar
```

LAB 9 | Artificial Intelligence

Aim : Knapsack problem by using Genetic Algorithm.

Code :

```
import random
import sys
import operator
import random as rd
from random import randint,randrange

class Knapsack(object):

    #initialize variables and lists
    def __init__(self):

        self.C = 0
        self.weights = []
        self.profits = []
        self.parents = []
        self.newparents = []
        self.bests = []
        self.best_p = []
        self.iterated = 1
        self.population = 0
        self.best_all = []

        # increase max recursion for long stack
        iMaxStackSize = 15000
        sys.setrecursionlimit(iMaxStackSize)

    # create the initial population
    def initialize(self,size):

        for i in range(self.population):
            parent = []
            for k in range(0, size):
                k = random.randint(0, 1)
                parent.append(k)
            self.parents.append(parent)

    # set the details of this problem
    def properties(self, weights, profits, C, population,size):

        self.weights = weights
```

```
    self.profits = profits
    self.C = C
    self.population = population
    self.initialize(size)

# calculate the fitness function of each list (sack)
def fitness(self, item):

    sum_w = 0
    sum_p = 0

    # get weights and profits
    for index, i in enumerate(item):
        if i == 0:
            continue
        else:
            sum_w += self.weights[index]
            sum_p += self.profits[index]

    # if greater than the optimal return -1 or the number otherwise
    if sum_w > self.C:
        return -1
    else:
        return sum_p

# run generations of GA
def evaluation(self):

    # loop through parents and calculate fitness
    best_pop = self.population // 2
    for i in range(len(self.parents)):
        parent = self.parents[i]
        ft = self.fitness(parent)
        self.bests.append((ft, parent))

    # sort the fitness list by fitness
    self.bests.sort(key=operator.itemgetter(0), reverse=True)
    self.best_p = self.bests[:best_pop]
    self.best_p = [x[1] for x in self.best_p]

# mutate children after certain condition
def mutation(self, ch):

    for i in range(len(ch)):
        k = random.uniform(0, 1)
        if k > 0.5:
            #if random float number greater than 0.5 flip 0 with 1 and vice versa
```

```
            if ch[i] == 1:
                ch[i] = 0
            else:
                ch[i] = 1
        return ch

    # crossover two parents to produce two children by mixing them under random ration each
time
    def crossover(self, ch1, ch2):

        threshold = random.randint(1, len(ch1)-1)
        tmp1 = ch1[threshold:]
        tmp2 = ch2[threshold:]
        ch1 = ch1[:threshold]
        ch2 = ch2[:threshold]
        ch1.extend(tmp2)
        ch2.extend(tmp1)

        return ch1, ch2

    # run the GA algorithm
    def run(self,num_gen):
        for gen in range(num_gen):
            # run the evaluation once
            self.evaluation()
            self.best_all.append((self.iterated,self.bests[0][0],self.bests[0][1]))
            newparents = []
            pop = len(self.best_p)-1

            # create a list with unique random integers
            sample = random.sample(range(pop), pop)
            for i in sample:
                # select the random index of best children to randomize the process
                if i < pop-1:
                    r1 = self.best_p[i]
                    r2 = self.best_p[i+1]
                    nchild1, nchild2 = self.crossover(r1, r2)
                    newparents.append(nchild1)
                    newparents.append(nchild2)
                else:
                    r1 = self.best_p[i]
                    r2 = self.best_p[0]
                    nchild1, nchild2 = self.crossover(r1, r2)
                    newparents.append(nchild1)
                    newparents.append(nchild2)

            # mutate the new children and potential parents to ensure global optima found
```

```
        for i in range(len(newparents)):
            newparents[i] = self.mutation(newparents[i])

        self.iterated += 1
        self.parents = newparents
        self.bests = []
        self.best_p = []

#define number of items
num_items = 4

#Capacity
C = 12

population = 16
number_generations = 15
weights = [5,3,7,2]
values = [12,5,10,7]
print('The list is as follows:')
print('Item No.   Weight   Value')
for i in range(num_items):
    print('{0}          {1}         {2}\n'.format(i+1, weights[i], values[i]))


k = Knapsack()
k.properties(weights, values, C, population, num_items)
k.run(number_generations)
fitness_history_max = [fitness[1] for fitness in k.best_all]
k.best_all.sort(key=operator.itemgetter(1), reverse=True)
print("Best Profit is {} at generation {} by selecting the tuples in the following order :
{}".format(k.best_all[0][1],k.best_all[0][0],k.best_all[0][2]))
```

Output :

```
This list is as follows:
Item No.        Weight          Value
1               5               12
2               3               5
3               7               10
4               2               7

Best Profit is 24 at generation 3 by selecting the tuples in the following order : [1, 1, 0, 1]
```

LAB 10| Artificial Intelligence

Aim: Travelling salesman problem using nearest neighbour heuristic and greedy edge heuristic.

- Using Nearest Neighbour Heuristic:

Code :

```
n = int(input("Enter No. of Nodes : "))

print("Enter Adjacency Matrix : ")
arr = [[0 for j in range(n)] for i in range(n)]
for i in range(0,n):
    arr[i] = list(map(int,input().split()))

print("Adjacency Matrix is : ")
print(arr)

for p in range(n):
    visited = [0 for l in range(n)]
    start=p
    cost=0
    j=1
    curr=start
    visited[start]=1
    print("Path For City " + str(curr+1))
    print(curr+1,end=' ')
    while j<n:
        nearest = arr[curr][curr]
        k = curr
        for i in range(n):
```

```
        if arr[curr][i]<nearest and visited[i]==0:
            nearest=arr[curr][i]
            k=i
    curr=k
    cost=cost+nearest
    visited[k]=1
    j=j+1
    print("> " + str(k+1),end=' ')
  cost=cost+arr[curr][start]
  print("")
  print("Cost for City " + str(p+1) + "=" + str(cost))
  print("------------------------------------")
```

Output :

```
Enter No. of Nodes : 6

Enter Adjacency Matrix :
999 10 20 30 40 50
10 999 31 21 51 41
20 31 999 12 59 100
30 21 12 999 5 8
40 51 59 5 999 69
50 41 100 8 69 999

Adjacency Matrix is :
[[999, 10, 20, 30, 40, 50],
 [10, 999, 31, 21, 51, 41],
 [20, 31, 999, 12, 59, 100],
[30, 21, 12, 999, 5, 8],
[40, 51, 59, 5, 999, 69],
[50, 41, 100, 8, 69, 999]]

Path For City 1
1 > 2 > 4 > 5 > 3 > 6
Cost for City 1=245
-----------------------------------
```

```
Path For City 2
2 > 1 > 3 > 4 > 5 > 6
Cost for City 2=157
-----------------------------------
Path For City 3
3 > 4 > 5 > 1 > 2 > 6
Cost for City 3=208
-----------------------------------
4 > 5 > 1 > 2 > 3 > 6
Cost for City 4=194
-----------------------------------
Path For City 5
5 > 4 > 6 > 2 > 1 > 3
Cost for City 5=143
-----------------------------------
Path For City 6
6 > 4 > 5 > 1 > 2 > 3
Cost for City 6=194
-----------------------------------
```

- Using Greedy Edge Heuristic :

Code:

```
n = int(input("Enter No. of Nodes : "))

print("Enter Adjacency Matrix : ")
arr = [[0 for j in range(n)] for i in range(n)]
for i in range(0,n):
    arr[i] = list(map(int,input().split()))

print("Adjacency Matrix is : ")
print(arr)
```

```
start=0
end=0
visited = [0 for l in range(n)]
cost=0
min=9999
for i in range(n):
    for j in range(n):
        if arr[i][j]<min:
            start=i
            end=j
            min=arr[i][j]
            cost=arr[i][j]

print(str(start+1) + " " + str(end+1))
c=2
visited[start]=1
visited[end]=1
while c<n:
    min_edge=9999
    node=0
    for j in range(n):
        if arr[start][j]<min_edge and visited[j]==0:
            node=j
            min_edge=arr[start][j]

    k=9999
    p=0
    for j in range(n):
        if arr[end][j]<k and visited[j]==0:
            p=j
            k=arr[end][j]
```

```
    if k<min_edge:
        print(str(end+1)+" " + str(p+1))
        cost=cost+k
        visited[p]=1
        end=p
    else:
        cost=cost+min_edge
        print(str(start+1)+" "+str(node+1))
        start=node
        visited[node]=1
    c=c+1

print(str(start+1)+" "+str(end+1))
cost=cost+arr[start][end]
print("Cost =",cost)
```

Output:

```
Enter No. of Nodes : 6

Enter Adjacency Matrix :
999 10 20 30 40 50
10 999 31 21 51 41
20 31 999 12 59 100
30 21 12 999 5 8
40 51 59 5 999 69
50 41 100 8 69 999

Adjacency Matrix is :
[[999, 10, 20, 30, 40, 50],
 [10, 999, 31, 21, 51, 41],
 [20, 31, 999, 12, 59, 100],
[30, 21, 12, 999, 5, 8],
 [40, 51, 59, 5, 999, 69],
 [50, 41, 100, 8, 69, 999]]

4 5
4 6
5 1
```

```
1 2
2 3
6 3
Cost = 194
```