**Deen Dayal Upadhyaya College**

**University of Delhi**



Artificial Intelligence Practicals

Submitted in Partial fulfilment in The Degree of Bachelor of Science Computer Science Honors

**Submitted To: Submitted By:**

Dr. Anuja Soni Vidhi Natani

(21HCS4191)

Q1. Write a prolog program to calculate the sum of two numbers.

**⇒ add() :-**

**write("Enter the first number : "),**

**read(X),**

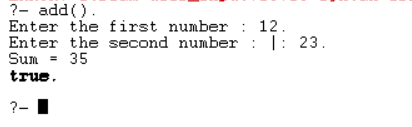
**write("Enter the second number : "),**

**read(Y),**

**Z is X+Y,**

**write("Sum = "), write(Z).**

***⇒ Output:***



Q2. Write a Prolog program to implement max(X, Y, M) so that M is the maximum of two numbers X and Y.

**⇒ max(X, Y, X) :- X>=Y.**

**max(X, Y, Y) :- Y>X.**

**maximum():-**

**write("Enter the first number : "),**

**read(X),**

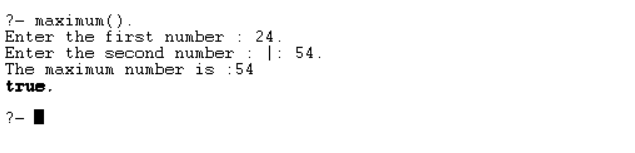
**write("Enter the second number : "),**

**read(Y),**

**max(X, Y, M),**

**write("The maximum number is :"), write(M).**

***⇒ Output:***



Q3. Write a program in PROLOG to implement factorial (N, F) where F represents the factorial of a number N.

**⇒ factorial(0, 1).**

**factorial(N, F) :-**

**N>0,**

**N1 is N-1,**

**factorial(N1, F1),**

**F is N\*F1.**

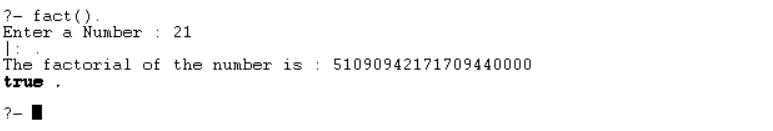
**fact():-**

**write("Enter a Number : "), read(X),**

**factorial(X, F),**

**write("The factorial of the number is : "), write(F).**

***⇒ Output:***



Q4. Write a program in PROLOG to implement generate\_fib(N,T) where T represents the Nth term of the fibonacci series.

**⇒ fib\_term(1, 0).**

**fib\_term(2, 1).**

**fib\_term(N, T) :-**

**N>0,**

**N1 is N-1,**

**N2 is N-2,**

**fib\_term(N1, T1),**

**fib\_term(N2, T2),**

**T is T1+T2.**

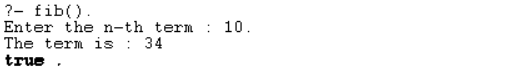
**fib():-**

**write("Enter the n-th term : "), read(X),**

**fib\_term(X, T),**

**write("The term is : "), write(T).**

***⇒ Output:***

****

Q5. Write a Prolog program to implement GCD of two numbers.

**⇒ gcd(X, 0, X) :- X>0.**

**gcd(X, Y, Z) :- Y>0,**

**Y1 is X mod Y,**

**gcd(Y, Y1, Z).**

**cgcd():-**

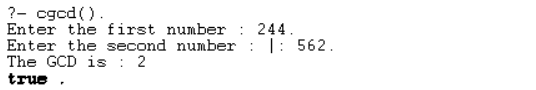
**write("Enter the first number : "), read(X),**

**write("Enter the second number : "), read(Y),**

**gcd(X,Y,Z),**

**write("The GCD is : "), write(Z).**

***⇒ Output:***

****

Q6. Write a Prolog program to implement power (Num,Pow, Ans) : where Num is raised to the power Pow to get Ans.

**⇒ pow(\_, 0, 1).**

**pow(N, 1, N).**

**pow(N, P, A) :-**

**P>1,**

**P1 is P-1,**

**pow(N, P1, A1),**

**A is N\*A1.**

**power():-**

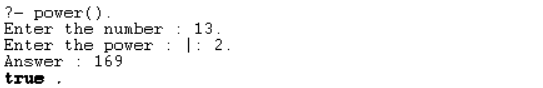
**write("Enter the number : "), read(X),**

**write("Enter the power : "), read(Y),**

**pow(X,Y,Z),**

**write("Answer : "), write(Z).**

***⇒ Output:***

****

Q7. Prolog program to implement multi (N1, N2, R) : where N1 and N2 denotes the numbers to be multiplied and R represents the result.

**⇒ multi(X, Y, Z) :-**

**Z is X\*Y.**

**multiply():-**

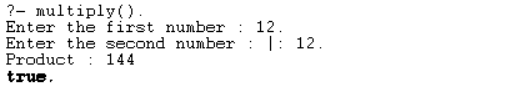
**write("Enter the first number : "), read(X),**

**write("Enter the second number : "), read(Y),**

**multi(X,Y,Z),**

**write("Product : "), write(Z).**

***⇒ Output:***

****

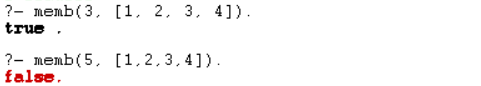
Q8. Write a Prolog program to implement memb(X, L): to check whether X is a member of L or not.

***⇒ % memb(X, L) checks if X is a member of the list L.***

**memb(X, [X | \_]).**

**memb(X, [\_ | Rest]) :- memb(X, Rest).**

***⇒ Output:***

****

Q9. Write a Prolog program to implement conc (L1, L2, L3) where L2 is the list to be appended with L1 to get the resulted list L3.

**⇒ *% Base case: concatenating an empty list with another list results in the other list***

**conc([], L, L).**

***% Recursive case: if the first list is not empty, we remove the head of the first list***

***% and append it to the result of concatenating the tail of the first list with the second list***

**conc([H|T1], L2, [H|T3]) :- conc(T1, L2, T3).**

***⇒ Output:***

****

Q10. Write a Prolog program to implement reverse (L, R) where List L is original and List R is reversed list.

***⇒ % Base case: the reverse of an empty list is the empty list***

**reverse([], []).**

**reverse([Head|Tail], Reversed) :-**

**reverse(Tail, ReversedTail),**

**append(ReversedTail, [Head], Reversed).**

***⇒ Output:***

****

Q11. Write a program in PROLOG to implement palindrome (L) which checks whether a list L is a palindrome or not.

***⇒ % Base case: An empty list is a palindrome.***

**palindrome([]).**

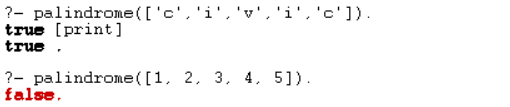
***% Recursive case: A list is a palindrome if its first and last elements are the same,***

***% and the sublist obtained by removing these elements is also a palindrome.***

**palindrome([\_]).**

**palindrome([H|T]) :- append(Middle, [H], T), palindrome(Middle).’**

**⇒ *Output:***

******

Q12. Write a Prolog program to implement sumlist(L, S) so that S is the sum of a given list L.

***⇒ % Base case: An empty list has a sum of 0.***

**sumlist([], 0).**

***% Recursive case: Sum the head of the list with the sum of the tail.***

**sumlist([Head|Tail], S) :-**

**sumlist(Tail, RestSum),**

**S is Head + RestSum.**

***⇒ Output:***

******

Q13. Write a Prolog program to implement two predicates evenlength(List) and oddlength(List) so that they are true if their argument is a list of even or odd length respectively.

***⇒ % Base case: An empty list has an even length.***

**evenlength([]).**

***% Recursive case: A list has an even length if its tail has an odd length.***

**evenlength([\_|T]) :- oddlength(T).**

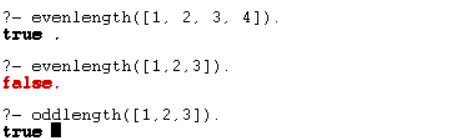
***% Base case: A list with a single element has an odd length.***

**oddlength([\_]).**

***% Recursive case: A list has an odd length if its tail has an even length.***

**oddlength([\_|T]) :- evenlength(T).**

***⇒ Output:***

****

Q14. Write a Prolog program to implement nth\_element (N, L, X) where N is the desired position, L is a list and X represents the Nth element of L.

***⇒****% nth\_element(N, L, X) succeeds if X is the Nth element of list L.*

**nth\_element(1, [X|\_], X).** *% Base case: X is the first element.*

**nth\_element(N, [\_|Rest], X) :-**

**N > 1,***% Ensure N is positive.*

**N1 is N - 1,** *% Decrement N.*

**nth\_element(N1, Rest, X).** *% Recurse on the rest of the list.*

*% Example usage:*

*% ?- nth\_element(3, [a, b, c, d, e], X).*

*% X = c*

***⇒Output:***

****

Q15. Write a Prolog program to implement maxlist(L, M) so that M is the maximum number in the list.

**⇒***% maxlist(L, M) succeeds if M is the maximum number in the list L.*

**maxlist([X], X).** *% Base case: Single-element list, M is that element.*

**maxlist([Head|Tail], Max) :-**

**maxlist(Tail, TailMax),** *% Recurse on the tail of the list.*

**(Head > TailMax -> Max = Head ; Max = TailMax).** *% Compare**Head with TailMax.*

*% Example usage:*

*% ?- maxlist([5, 8, 3, 12, 7], Max).*

*% Max = 12*

***⇒Output:***

******

Q16. Write a prolog program to implement insert\_nth (I, N, L, R) that inserts an item I into Nth position of list L to generate a list R.

**⇒** *% insert\_nth(I, N, L, R) succeeds if R is the list obtained by inserting I at position N in L.*

**insert\_nth(I, 1, L, [I|L]).** *% Base case: Insert I at the beginning.*

**insert\_nth(I, N, [X|Rest], [X|R]) :-**

**N > 1,** *% Ensure N is positive.*

**N1 is N - 1,** *% Decrement N.*

**insert\_nth(I, N1, Rest, R).** *% Recurse on the rest of the list.*

***⇒Output:***

******

Q17. Write a Prolog program to implement delete\_nth (N, L, R) that removes the element on Nth position from a list L to generate a list R.

**⇒** *% delete\_nth(N, L, R) succeeds if R is the list obtained by removing the Nth element from L.*

**delete\_nth(1, [\_|Tail], Tail).** *% Base case: Remove the first element.*

**delete\_nth(N, [X|Rest], [X|R]) :-**

**N > 1,** *% Ensure N is positive.*

**N1 is N - 1,** *% Decrement N.*

**delete\_nth(N1, Rest, R).** *% Recurse on the rest of the list.*

***⇒ Output:***

******

Q18. Write a program in PROLOG to implement merge (L1, L2, L3) where L1 is first ordered list and L2 is second ordered list and L3 represents the merged list.

**⇒** *% Base case: If both lists are empty, the merged list is also empty.*

**merge([], [], []).**

*% If L1 is empty, L3 is simply L2.*

**merge([], L2, L2).**

*% If L2 is empty, L3 is simply L1.*

**merge(L1, [], L1).**

*% Merge the heads of L1 and L2 into L3.*

**merge([X | Rest1], [Y | Rest2], [X | Rest3]) :-**

**X =< Y,** *% X is less than or equal to Y*

**merge(Rest1, [Y | Rest2], Rest3).**

**merge([X | Rest1], [Y | Rest2], [Y | Rest3]) :-**

**X > Y,** *% X is greater than Y*

**merge([X | Rest1], Rest2, Rest3).**

***⇒Output:***

******