|  |
| --- |
| /\***Practical 1** |
|  | Write a prolog program to calculate the sum of two numbers.\*/ |
|  | sum:- |
|  | write("Enter First number: "), |
|  | read(X), |
|  | write("Enter Second number"), |
|  | read(Y), |
|  | Z is X+Y, |
|  | write("Sum of "),write(X), |
|  | write(" and "),write(Y), |
|  | write(" is "),write(Z), |
|  | nl, cont. |
|  |  |
|  | cont:- |
|  | write("Do you want to do some more sum(y/n)? "), |
|  | read(X), |
|  | check(X). |
|  |  |
|  | check(y):- |
|  | sum. |
|  |  |
|  | check(n):- |
|  | !. |
|  |  |
|  | check(X):- |
|  | write("Please choose right choice "), |
|  | nl,cont. |
|  |  |
|  | /\***Practical 2** |
|  | Write a Prolog program to implement max(X, Y, M) so that M is the maximum of two numbers X and Y.\*/ |
|  | max:- |
|  | write("Enter First number: "), |
|  | read(X), |
|  | write("Enter Second number"), |
|  | read(Y), |
|  | ( X =< Y |
|  | -> Z = Y |
|  | ; Z = X |
|  | ),write("Maximum between "),write(X), |
|  | write(" and "),write(Y), |
|  | write(" is "),write(Z), |
|  | !. |
|  |  |
|  | /\***Practical 3** |
|  | 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,Result1), |
|  | F is Result1\*N. |
|  | factorial:- |
|  | write("Enter number to find factorial: "), |
|  | read(X), |
|  | factorial(X,F), |
|  | write("The factorial of "),write(X), |
|  | write(" is: "),write(F),!. |
|  |  |
|  |  |
|  | /\***Practical 4** |
|  | Write a program in PROLOG to implement generate\_fib(N,T) where T represents theNth term of the fibonacci series.\*/ |
|  | generate\_fib(1,0). |
|  | generate\_fib(2,1). |
|  | generate\_fib(N,T):- |
|  | N1 is N-1, |
|  | generate\_fib(N1,T1), |
|  | N2 is N-2, |
|  | generate\_fib(N2,T2), |
|  | T is T1+T2. |
|  | fibonacci:- |
|  | write("Enter the fibonacci term you want: "), |
|  | read(X), |
|  | generate\_fib(X,F), |
|  | write("The number at "),write(X), |
|  | write(" in fibonacci series is: "),write(F),!. |
|  |  |
|  | /\***Practical 5** |
|  | Write a Prolog program to implement GCD of two numbers.\*/ |
|  | gcd(0, X, X):- |
|  | X > 0, !. |
|  | gcd(X, Y, R):- |
|  | X >= Y, X1 is X-Y, gcd(X1,Y,R). |
|  | gcd(X, Y, R):- |
|  | X < Y, X1 is Y-X, gcd(X1,X,R). |
|  | gcd:- |
|  | write("Enter first number: "), |
|  | read(X), |
|  | write("Enter second number: "), |
|  | read(Y), |
|  | gcd(X,Y,F), |
|  | write("GCD of "),write(X),write(" and "), |
|  | write(Y),write(" is: "),write(F),!. |
|  |  |
|  |  |
|  | /\***Practical 6** |
|  | Write a Prolog program to implement power (Num,Pow, Ans) : where Num is raised to the power Pow to get Ans.\*/ |
|  | power(\_,0,1). |
|  | power(Num,Pow,Ans) :- Pow1 is Pow - 1, |
|  | pow(Num,Pow1,Ans1), Ans is Ans1\*Num. |
|  | power:- |
|  | write("Enter the base: "), |
|  | read(X), |
|  | write("Enter the exponent: "), |
|  | read(Y), |
|  | power(X,Y,F), |
|  | write("The result of "),write(X), |
|  | write(" to the power "),write(Y), |
|  | write(" is: "),write(F),!. |
|  |  |
|  | /\***Practical 7** |
|  | Prolog program to implement multi (N1, N2, R) : where N1 and N2 denotes the numbers to be multiplied and R represents the result.\*/ |
|  | multi:- |
|  | write("Enter first Number: "), |
|  | read(X), |
|  | write("Enter second Number: "), |
|  | read(Y), |
|  | R is X\*Y, |
|  | write("Product of "),write(X), |
|  | write(" and "),write(Y), |
|  | write(" is: "),write(R),!. |
|  |
| /\***Practical 8** |
|  | Write a Prolog program to implement memb(X, L): to check whether X is a member of L or not.\*/ |
|  | memb(X,[X|T]). |
|  | memb(X,[H|T]):- |
|  | memb(X,T). |
|  | member:- |
|  | write("Enter the List: "), |
|  | read(L), |
|  | write("Enter the number to be found: "), |
|  | read(X), |
|  | write(X), |
|  | (memb(X,L)-> |
|  | write(" is the member of the given list") |
|  | ;write(" is not the member of the given list")),!. |
|  |  |
|  | /\***Practical 9** |
|  | 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.\*/ |
|  | conc([],L,L). |
|  | conc([X|L1],L2,[X|L3]):- |
|  | conc(L1,L2,L3). |
|  | concatenate:- |
|  | write("Enter First List: "), |
|  | read(L1), |
|  | write("Enter Second List: "), |
|  | read(L2), |
|  | conc(L1,L2,L3), |
|  | write("The concatenated list is: "), |
|  | write(L3),!. |
|  |  |
|  | /\***Practical 10** |
|  | Write a Prolog program to implement reverse (L, R) where List L is original and List R is reversed list.\*/ |
|  | reverse([],[]). |
|  | reverse([X|L1],L2):- |
|  | reverse(L1,L3), |
|  | conc(L3,[X],L2). |
|  | reverse:- |
|  | write("Enter the list to be reversed: "), |
|  | read(L1), |
|  | reverse(L1,L2), |
|  | write("Reversed list is: "), |
|  | write(L2),!. |
|  |  |