deciToBin(0):- write(0).

deciToBin(1):- write(1).

deciToBin(N):-

N>1,

Bit is N mod 2,

Rest is N//2,

deciToBin(Rest),

write(Bit).

%1 deciToBin(230).

11100110

deciToHex(0) :- write(0).

deciToHex(N) :-

N > 0,

deciToHexHelper(N).

deciToHexHelper(0).

deciToHexHelper(N) :-

N > 0,

Remainder is N mod 16,

Quotient is N // 16,

deciToHexHelper(Quotient),

writeHexDigit(Remainder).

writeHexDigit(D) :-

D < 10, write(D).

writeHexDigit(10) :- write('A').

writeHexDigit(11) :- write('B').

writeHexDigit(12) :- write('C').

writeHexDigit(13) :- write('D').

writeHexDigit(14) :- write('E').

writeHexDigit(15) :- write('F').

%2 deciToHex(76).

04C

hex\_digit(0,'0').

hex\_digit(1,'1').

hex\_digit(2,'2').

hex\_digit(3,'3').

hex\_digit(4,'4').

hex\_digit(5,'5').

hex\_digit(6,'6').

hex\_digit(7,'7').

hex\_digit(8,'8').

hex\_digit(9,'9').

hex\_digit(10,'A').

hex\_digit(11,'B').

hex\_digit(12,'C').

hex\_digit(13,'D').

hex\_digit(14,'E').

hex\_digit(15,'F').

deciToHex(0):- write('0').

deciToHex(N):-

N>0,

Remainder is N mod 16,

Rest is N // 16,

deciToHex(Rest),

hex\_digit(Remainder, Digit),

write(Digit).

%deciToHex(76).

04C

%factorial

factorial(0,1).

factorial(N, Res):-

N>0,

N1 is N-1,

factorial(N1, TempRes),

Res is N \*TempRes.

% ?- factorial(5, Res).

%fibonacci(N)

%fibonacci(N)

fibonacci(N):- fib\_helper(0,1,N).

fib\_helper(A,B,N):-

A =< N,

write(A), write(‘),

Next is A+B,

fib\_helper(B, Next, N).

% ?- fibonacci(30).

0‘1‘1‘2‘3‘5‘8‘13‘21

displayDigits(N) :-

N < 10,

write(N).

displayDigits(N) :-

N >= 10,

Digit is N mod 10,

Rest is N // 10,

displayDigits(Rest),

write(' '),

write(Digit).

%displayDigits(12345).

%?-

countOfDigits(N, 1) :-

N < 10.

countOfDigits(N, Res) :-

N >= 10,

Rest is N // 10,

countOfDigits(Rest, SubRes),

Res is SubRes + 1.

% countOfDigits(1234645, Res).

%sum of digits

sumOfDigits(N, N) :-

N < 10.

% Recursive case: process multi-digit number

sumOfDigits(N, Res) :-

N >= 10,

Digit is N mod 10, % Get the last digit

Rest is N // 10, % Remove the last digit

sumOfDigits(Rest, SubRes), % Sum digits in the rest

Res is SubRes + Digit.

sumOfDigits(1234645, Res).

%reverse a number

reverseNumberHelper(0, Acc, Acc).

reverseNumberHelper(N, Acc, Res) :-

N > 0,

Digit is N mod 10,

NewAcc is Acc \* 10 + Digit,

Rest is N // 10,

reverseNumberHelper(Rest, NewAcc, Res).

reverseNumber(N, Res) :-

reverseNumberHelper(N, 0, Res).

%?- reverseNumber(56467, Res).

% GCD

gcd(A, 0):- write(A).

gcd(A, B):-

B>0,

R is A mod B,

gcd(B, R).

%?- gcd(4,8).

%armstrongNumber upto N

isArmstrong(N) :-

number\_chars(N, Digits),

length(Digits, Len),

maplist(atom\_number, Digits, DList),

maplist(power(Len), DList, Powers),

sum\_list(Powers, N).

power(P, D, R) :- R is D\*\*P.

armstrongNumbers(N) :-

between(1, N, X),

isArmstrong(X),

write(X), nl,

fail.

armstrongNumbers(\_).

%?- armstrongNumbers(500).

% checkPrime(N).

checkPrime(1) :- write('Not Prime'), !, fail.

checkPrime(N) :- N > 1, \+ hasDivisor(N, 2), write('Prime'), !.

checkPrime(\_) :- write('Not Prime').

hasDivisor(N, D) :- D \* D =< N, (N mod D =:= 0; hasDivisor(N, D + 1)).

% display primeNumbers(N).

primeNumbers(N) :-

between(2, N, X),

checkPrime(X),

write(X), nl,

fail.

primeNumbers(\_).

checkPrime(1) :- !, fail.

checkPrime(N) :- N > 1, \+ hasDivisor(N, 2).

hasDivisor(N, D) :- D \* D =< N, (N mod D =:= 0; hasDivisor(N, D + 1)).

% primeNumbers(Low, High).

primeNumbers(Low, High) :-

between(Low, High, X),

checkPrime(X),

write(X), nl,

fail.

primeNumbers(\_, \_).

checkPrime(1) :- !, fail.

checkPrime(N) :- N > 1, \+ hasDivisor(N, 2).

hasDivisor(N, D) :- D \* D =< N, (N mod D =:= 0; hasDivisor(N, D + 1)).

% twinPrime(N).

twinPrime(N) :-

between(2, N, X),

checkPrime(X),

Twin is X + 2,

Twin =< N,

checkPrime(Twin),

write('('), write(X), write(', '), write(Twin), write(')'), nl,

fail.

twinPrime(\_).

checkPrime(1) :- !, fail.

checkPrime(N) :- N > 1, \+ hasDivisor(N, 2).

hasDivisor(N, D) :- D \* D =< N, (N mod D =:= 0; hasDivisor(N, D + 1)).

?- twinPrime(20).

(3, 5)

(5, 7)

(11, 13)

(17, 19)

true.

?- twinPrime(50).

(3, 5)

(5, 7)

(11, 13)

(17, 19)

(29, 31)

(41, 43)

true.

% display list

displayList([]). % Base case: empty list

displayList([H|T]) :-

write(H), nl, % Write the head of the list

displayList(T). % Recursively process the tail

% reverse display of a list

reverseDisplayList([]). % Base case: empty list

reverseDisplayList([H|T]) :-

reverseDisplayList(T), % Recursively process the tail

write(H), nl. % Write the head after the recursive call

?- reverseDisplayList([1, 2, 3, 4, 5]).

5

4

3

2

1

true.

?- reverseDisplayList([apple, banana, cherry]).

cherry

banana

apple

true.

listLength([], 0). % Base case: the length of an empty list is 0

listLength([\_|T], Length) :-

listLength(T, SubLength), % Recursively find the length of the tail

Length is SubLength + 1. % Add 1 for the head

?- listLength([apple, banana, cherry], Length).

Length = 3.

displayFirstN(\_, 0, []). % Base case: when N is 0, the result is an empty list

displayFirstN([H|T], N, [H|Result]) :-

N > 0,

N1 is N - 1,

displayFirstN(T, N1, Result). % Recursively process the tail

?- displayFirstN([a, b, c, d, e, f], 3, Result).

Result = [a, b, c].

?- displayFirstN([1, 2, 3, 4, 5], 2, Result).

Result = [1, 2].

displayLastN(List, N, Result) :-

length(List, Len),

Start is Len - N,

Start >= 0,

skip(List, Start, Result).

skip(List, 0, List). % Base case: when Start is 0, the remaining list is the result

skip([\_|T], N, Result) :-

N > 0,

N1 is N - 1,

skip(T, N1, Result). % Skip the first N elements

?- displayFirstN([a, b, c, d, e, f], 3, Result).

Result = [a, b, c].

?- displayFirstN([1, 2, 3, 4, 5], 2, Result).

Result = [1, 2].

displayNthToMth(\_, N, M, []) :- N > M. % Base case: if N > M, return an empty list

displayNthToMth([\_|T], N, M, Result) :-

N > 1,

N1 is N - 1,

M1 is M - 1,

displayNthToMth(T, N1, M1, Result).

displayNthToMth([H|T], 1, M, [H|Result]) :-

M > 0,

M1 is M - 1,

displayNthToMth(T, 1, M1, Result).

?- displayNthToMth([a, b, c, d, e, f], 2, 4, Result).

Result = [b, c, d].

?- displayNthToMth([1, 2, 3, 4, 5, 6], 3, 5, Result).

Result = [3, 4, 5].

insertAtHead(Element, List, [Element|List]).

?- insertAtHead(a, [b, c, d], Result).

Result = [a, b, c, d].

?- insertAtHead(1, [2, 3, 4], Result).

Result = [1, 2, 3, 4].

appendLists([], L2, L2). % Base case: appending an empty list results in the second list

appendLists([H|T], L2, [H|Result]) :-

appendLists(T, L2, Result). % Recursively append the tail of the first list

displayAppendedLists(L1, L2) :-

appendLists(L1, L2, Result),

write(Result).

?- appendLists([a, b, c], [d, e, f], Result).

Result = [a, b, c, d, e, f].

?- displayAppendedLists([1, 2, 3], [4, 5, 6]).

[1, 2, 3, 4, 5, 6]

true.

sumList([], 0).

sumList([H|T], Sum) :- sumList(T, SubSum), Sum is H + SubSum.

displaySum(List) :- sumList(List, Sum), write(Sum).

?- displaySum([1, 2, 3, 4, 5]).

15

true.

?- sumList([10, 20, 30], Sum).

Sum = 60.

searchElement(X, [X|\_]) :- write('Element found'), !.

searchElement(X, [\_|T]) :- searchElement(X, T).

searchElement(\_, []) :- write('Element not found').

?- searchElement(3, [1, 2, 3, 4, 5]).

Element found

true.

?- searchElement(6, [1, 2, 3, 4, 5]).

Element not found

true.

deleteElement(\_, [], []). % Base case: empty list

deleteElement(X, [X|T], T). % If the head matches the element, skip it

deleteElement(X, [H|T], [H|Result]) :- deleteElement(X, T, Result).

?- deleteElement(3, [1, 2, 3, 4, 5], Result).

Result = [1, 2, 4, 5].

?- deleteElement(6, [1, 2, 3, 4, 5], Result).

Result = [1, 2, 3, 4, 5].

%DFS

edge(s, a).

edge(s, b).

edge(s, c).

edge(a, d).

edge(c, e).

edge(e, f).

edge(f, g).

edge(g, h).

connected(X, Y) :-

edge(X, Y); edge(Y, X).

dfs(\_, [], \_, \_) :-

write('not found'), !.

dfs(Goal, [Goal|\_], CLOSED, Result) :-

write('found'),

append(CLOSED, [Goal], Path),

Result = Path, !.

dfs(Goal, [N|Rest], CLOSED, Result) :-

OPEN = Rest,

findall(X, (connected(N, X), not(member(X, OPEN)), not(member(X, CLOSED))), M),

append(CLOSED, [N], CLOSEDM),

append(M, OPEN, OPENM), % For DFS, prepend M to OPEN

dfs(Goal, OPENM, CLOSEDM, Result).

?- dfs(h, [s], [], Result).

found

Result = [s, c, e, f, g, h].

%BFS

edge(s,a).

edge(s,b).

edge(s,c).

edge(a,d).

edge(c,e).

edge(e,f).

edge(f,g).

edge(g,h).

connected(X, Y):-

edge(X,Y);edge(Y,X).

bfs([],\_):-write('not found'),!.

bfs(Goal, [Goal|\_], CLOSED, Result):-write('found'),append(CLOSED,[Goal],Path),Result=Path,!.

bfs(Goal,[N|Rest],CLOSED,Result):-

OPEN=Rest,

findall(X, (connected(N, X),not(member(X, OPEN)),not(member(X, CLOSED))), M),

append(CLOSED,[N],CLOSEDM),

append(OPEN, M, OPENM), % for DFS append(M, OPEN, OPENM)

bfs(Goal,OPENM,CLOSEDM,Result).

bfs(h, [s], [], Result).