Recall that a non-negative integer $n$ can be encoded (in unary) as $n$ succ’s applied to 0

\[
\text{numeral}(0).
\text{numeral}(\text{succ}(X)) :- \text{numeral}(X).
\]

In general, Prolog terms constructed from a constant null and $k + 1$ unary functors $f_0, \ldots, f_k$ amount to strings over an alphabet $\{a_0, \ldots, a_k\}$, with null encoding the empty string, and $f_i(\text{null})$ encoding the string $a_i$. For strings of length $> 1$, it will prove useful (for some purposes such as the arithmetic encoding below) to encode the string in reverse, representing, for example, $a_2a_3$ as $f_3(f_2(\text{null}))$, rather than $f_2(f_3(\text{null}))$.

To simplify notation, let us work with the alphabet $\{0, 1\}$ (with $k = 1, a_0 = 0, a_1 = 1$) and unary functors $f_0, f_1$. Let “pterm” abbreviate “Prolog term built from null, $f_0, f_1$” as described by the clauses

\[
\text{pterm}(\text{null}).
\text{pterm}(f_0(X)) :- \text{pterm}(X).
\text{pterm}(f_1(X)) :- \text{pterm}(X).
\]

Putting numbers in binary form,

(†) 0 becomes the bitstring 0 and pterm $f_0(\text{null})$,
1 becomes the bitstring 1 and pterm $f_1(\text{null})$,
2 becomes the bitstring 10 and pterm $f_0(f_1(\text{null}))$,
3 becomes the bitstring 11 and pterm $f_1(f_1(\text{null}))$,
\[\vdots\]

Note that pterms such as null and $f_1(f_0(f_0(\text{null})))$ are excluded from (†), even though we can associate non-negative integers (0 and 3) with them.

N.B. In solving the problems below, you are banned from using built-in arithmetic predicates or lists in Prolog.
Problem 1. Define a predicate \texttt{incr(P1,P2)} over pterms \texttt{P1} and \texttt{P2} such that under (†), \texttt{P2} is the successor of \texttt{P1}. For example, as 3 is 2+1,

\begin{verbatim}
| ?- incr(f0(f1(null)),X).
  X = f1(f1(null)) ;
  no
\end{verbatim}

You are free to define \texttt{incr} such that \texttt{incr(null,X)} holds for no \texttt{X} or else only for \texttt{X=f1(null)}. Likewise for \texttt{incr(P,X)} where \texttt{P} is some other pterm not in (†).

Problem 2. Define a predicate \texttt{legal(P)} true exactly of pterms \texttt{P} mentioned by (†). Hence,

\begin{verbatim}
| ?- legal(X).
  X = f0(null) ;
  X = f1(null) ;
  X = f0(f1(null)) ;
  X = f1(f1(null)) ;
  X = f0(f0(f1(null))) ;
  X = f1(f0(f1(null))) ;
  ...
\end{verbatim}

Using \texttt{legal}, revise your predicate \texttt{incr} to \texttt{incrR} such that

\begin{verbatim}
?- incrR(X,Y).
  X = f0(null), Y = f1(null) ;
  X = f1(null), Y = f0(f1(null)) ;
  X = f0(f1(null)), Y = f1(f1(null)) ;
  X = f1(f1(null)), Y = f0(f0(f1(null))) ;
  X = f0(f0(f1(null))), Y = f1(f0(f1(null))) ;
  X = f1(f0(f1(null))), Y = f0(f1(f1(null))) ;
  ...
\end{verbatim}

Problem 3. Define a predicate \texttt{add(P1,P2,P3)} over pterms \texttt{P1,P2} and \texttt{P3} such that under (†), \texttt{P3} is \texttt{P1} plus \texttt{P2}. For example, as 3 is 1+2,

\begin{verbatim}
| ?- add(f1(null),f0(f1(null)),X).
  X = f1(f1(null)) ;
  no
\end{verbatim}
Problem 4. Define a predicate \texttt{mult(P1,P2,P3)} over pterms \texttt{P1,P2} and \texttt{P3} such that under (†), \texttt{P3} is \texttt{P1} times \texttt{P2}. For example, as 2 is \(1 \times 2\),

\[
| \texttt{?- mult(f1(null),f0(f1(null)),X).} \\
X = f0(f1(null)); \\
\text{no}
\]

Problem 5. Define a predicate \texttt{revers(P, RevP)} that takes a pterm \texttt{P} and reverses it to \texttt{RevP} so that, for example,

\[
| \texttt{?- revers(f0(f1(null)),X).} \\
X = f1(f0(null)); \\
\text{no}
\]

Problem 6. Define a predicate \texttt{normalize(P, Pn)} true of pterms \texttt{P} and \texttt{Pn} such that \texttt{legal(Pn)} and \texttt{P} and \texttt{Pn} encode the same number, \(\texttt{enc(P)} = \texttt{enc(Pn)}\), where

\[
\begin{align*}
\texttt{enc(null)} & := 0 \\
\texttt{enc(f0(X))} & := 2 \times \texttt{enc(X)} \\
\texttt{enc(f1(X))} & := 2 \times \texttt{enc(X)} + 1
\end{align*}
\]

For example,

\[
| \texttt{?- normalize(null, X).} \\
X = f0(null); \\
\text{no} \\
| \texttt{?- normalize(f1(f0(f0(null))), X).} \\
X = f1(null); \\
\text{no}
\]

Feel free to use Prolog’s built-in binary predicate \(\neq\) for inequality (e.g. \texttt{null \neq f0(null)}).

Final note. To help your demonstrator (and yourself) test your solutions, please add the following clauses to your Prolog code.

\[
\% \text{ test add inputting numbers N1 and N2} \\
\text{testAdd(N1,N2,T1,T2,Sum,SumT)} :- \text{numb2pterm(N1,T1), numb2pterm(N2,T2),} \\
\text{add(T1,T2,SumT), pterm2numb(SumT,Sum)}.
\]
% test mult inputting numbers N1 and N2
testMult(N1,N2,T1,T2,N1N2,T1T2) :- numb2pterm(N1,T1), numb2pterm(N2,T2),
    mult(T1,T2,T1T2), pterm2numb(T1T2,N1N2).

% test revers inputting list L
testRev(L,Lr,T,Tr) :- ptermlist(T,L), revers(T,Tr), ptermlist(Tr,Lr).

% test normalize inputting list L
testNorm(L,T,Tn,Ln) :- ptermlist(T,L), normalize(T,Tn), ptermlist(Tn,Ln).

% make a pterm T from a number N    numb2term(+N,?T)
numb2pterm(0,f0(null)).
numb2pterm(N,T) :- N>0, M is N-1, numb2pterm(M,Temp), incr(Temp,T).

% make a number N from a pterm T    pterm2numb(+T,?N)
pterm2numb(null,0).
pterm2numb(f0(X),N) :- pterm2numb(X,M), N is 2*M.
pterm2numb(f1(X),N) :- pterm2numb(X,M), N is 2*M +1.

% reversible ptermlist(T,L)
ptermlist(null,[]).
ptermlist(f0(X),[0|L]) :- ptermlist(X,L).
ptermlist(f1(X),[1|L]) :- ptermlist(X,L).

Apart from these clauses, your program should make no use of Prolog’s built-in arithmetic and list predicates.