1. Să se scrie funcţia recursivă corespunzătoare pentru funcţia Ackerman definită astfel:

$$A(m,x)=\begin{cases} x+1, m=0 \\ A(m-1,1), x=0 \\ A(m-1,A(m,x-1)), altfel \end{cases}$$

```
% main.pro
implement main
    open core
class predicates
    ackermann : (integer M, integer N, integer A) procedure (i,i,o).
clauses
        ackermann(0,N,A) :-    !,
        A = N+1.
    ackermann(M,0,A) :-        !,
        ackermann(M-1,1,A1),
        A=A1.
    ackermann(M,N,A) :-
        ackermann(M,N-1,A1),
        ackermann(M-1,A1,A2),
        A=A2.
clauses
    run():-
        console::init(),
        ackermann(2,2,A),
        stdio::writef("Ack(2,2)=%\n",A),
        succeed().
end implement main
goal
    mainExe::run(main::run).
```

2. Să se scrie funcţia recursivă corespunzătoare pentru funcţia cmmdc a două numere a şi b definită astfel:

$$cmmdc(a,b)=\begin{cases} a, a=b \\ cmmdc(a-b,b), a>b \\ cmmdc(a,b-a), a<b \end{cases}$$

Procedural:

```
% main.pro
implement main
    open core
class predicates
    cmmdc : (integer N, integer M, integer CMMDC) procedure (i,i,o).
clauses
    cmmdc(N,M,CMMDC) :-    if N=M    then     CMMDC = N
    elseif N>M    then          N1=N-M,
```

```
        cmmdc(N1,M,C),          CMMDC = C
    else
      M1=M-N,
      cmmdc(N,M1,C),       CMMDC=C
    end if.
clauses
    run():-
      console::init(),
      cmmdc(120,28,CMMDC),
      stdio::writef("cmmdc(120,28)=%", CMMDC),
      succeed().
end implement main
goal
    mainExe::run(main::run).
```

Prolog:

```
implement main
    open core
class predicates
    cmmdc : (integer A, integer B, integer C) nondeterm (i,i,o).
clauses
        cmmdc(A, A, C):-          C=A.
        cmmdc(A, B, C):-          A>B,
          cmmdc(A-B,B,C).
          cmmdc(A, B, C):-          A<B,
          cmmdc(A,B-A,C).
        clauses
    run():-
      console::init(),
       cmmdc(28,120,C),
      stdio::writef("cmmdc=%",C),
      !,
      succeed().
    run().
end implement main
goal
    mainExe::run(main::run).
```

3. Să se scrie funcţia recursivă corespunzătoare pentru functia combinări definită astfel:

$$c(n,k)=\begin{cases}1, k=n\\ 1, k=0\\ c(n-1,k)+c(n-1,k-1)\end{cases}$$

Procedural:

```
% main.pro
implement main
   open core
class predicates
   combinari : (integer N, integer K, integer C) procedure (i,i,o).
clauses
   combinari(N,K,C) :-
      stdio::writef("combinari de % luate cate % \n",N,K),
      if K=0   then          C=1
      elseif  K=N    then          C=1
         else
            N1=N-1,
            K1=K-1,
            combinari(N1,K,C1),
            combinari(N1,K1,C2),
            C=C1+C2
         end if.
clauses
   run():-
      console::init(),
      combinari(3,2,C),
      stdio::writef("combinari de % luate cate % = %\n",3,2,C),
      succeed().
end implement main
goal
   mainExe::run(main::run).
```

Prolog:

```
implement main
   open core
class predicates
   combinari : (integer N, integer K, integer C) procedure (i,i,o).
clauses
       combinari(_, 0, C):-          !,
         C=1.
       combinari(N, N, C):-          !,
         C=1.
       combinari(N,K,C) :-         N1=N-1,
         K1=K-1,
         combinari(N1, K, C1),
```

```
            combinari(N1, K1, C2),
            C=C1+C2.
        clauses
    run():-
        console::init(),
        N=5,
        K=4,
         combinari(N,K,C),
        stdio::writef("combinari de % luate cate % = %\n",N,K,C),
        succeed().
end implement main
goal
    mainExe::run(main::run).
```

4. Să se scrie funcţia recursivă corespunzătoare pentru funcţia k^n

```
% main.pro
implement main
    open core
class predicates
    putere : (integer K, integer N, integer P) procedure (i,i,o).
clauses
    putere(K,0,P) :-      !,
        P = 1,
        stdio::writef("%^0 = 1\n",K).
    putere(K,N,P) :-
    putere(K,N-1,P1),
        P = K*P1,
        stdio::writef("%^% = %\n", K,N,P).
clauses
    run():-
        console::init(),
        putere(3,4,P),
        succeed().
end implement main
goal
    mainExe::run(main::run).
```