2014/15

Zad. 1
Dany jest fragment bazy faktów:

ksiazka(16, 'Flatlandia', autor('Edvin Abbot', 1838-1926), wydanie('Sell & Co', 1884)).
ksiazka(28, 'R. U. R.', autor('Karel Capek', 1890-1938), wydanie('Palyers Press', 2002)).
ksiazka(34, 'Kobieta z wydm', autor('Kobo Abe', 1924-1993), wydanie('Wydawnictwo Znak', 2007)).
ksiazka(36, 'Zamek', autor('Frans Kafka', 1883-1924), wydanie('Zielona Sowa', 2001)).
ksiazka(37, 'Gargantua i Pantagruel', autor('Francois Rabelais', 1494-1553), wydanie('Siedmiorόg', 2004)).

Należy napisać klauzule, które znajdują się w bazie:

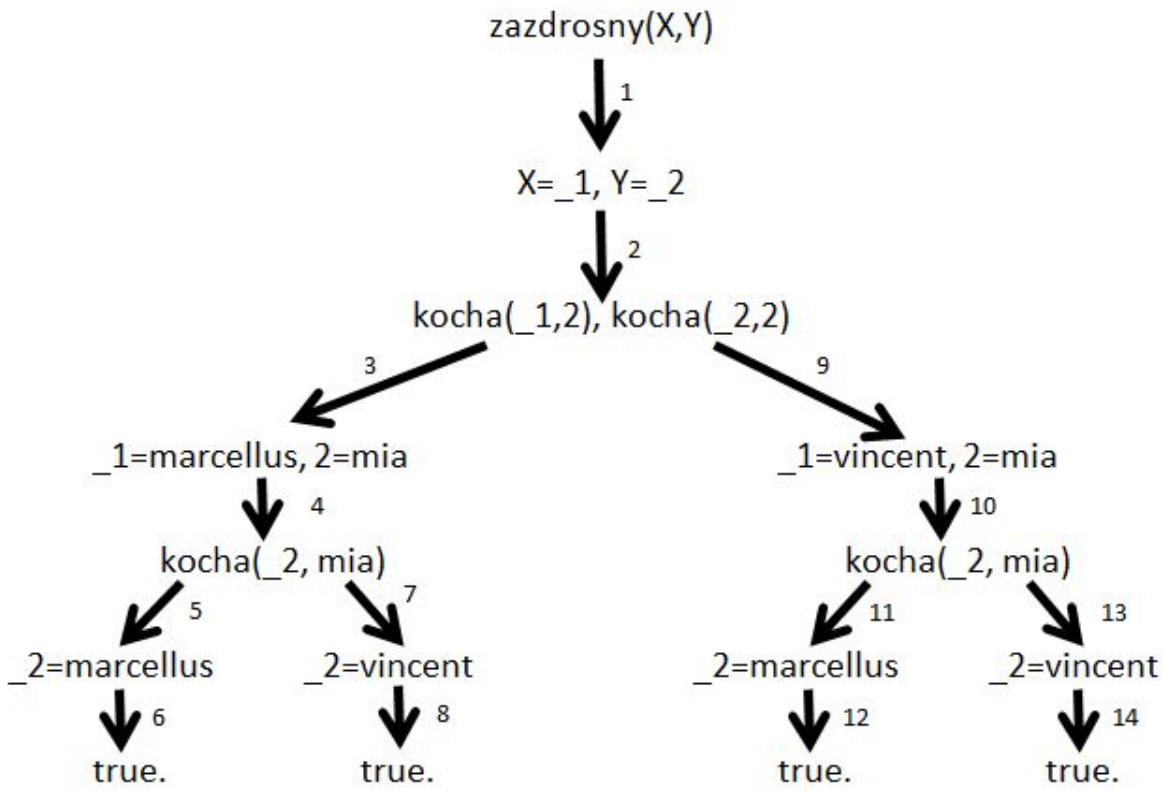
1. **książki wydane po śmierci swojego autora**
2. **autorów, którzy mieli szanse się spotkać (żyli w tym samym czasie)**
3. **listę wszystkich autorów żyjących (należy zastosować predykaty agregacyjne)**

```
poSmiertnie(Tytul) :-
        ksiazka(_,Tytul, autor(_,_-RokSmierci), wydanie(_, RokWydania)),
        RokSmierci @< RokWydania.

mogliSpotkac(Autor1, Autor2) :-
        ksiazka(_, _, autor(Autor1,RokUrodzenia1-RokSmierci1),_),
        ksiazka(_, _, autor(Autor2,RokUrodzenia2-RokSmierci2),_),
        Autor1 @< Autor2, %zapewnia ulozenie alfabetyczne, wiec nie powtorza sie np. X, Y ; Y, X
        RokUrodzenia1 @< RokSmierci2, %operator @< moze porownywac liczby i slowa, w tym przypadku nie jest koniecznie potrzebny, ale w linijce Autor1 @< Autor2 juz tak (IIRC)
        RokUrodzenia2 @< RokSmierci1.

autorzyZyjacy(X) :- findall(Z,(ksiazka(_,_,autor(Z,_-D),_), D >= 2016),X).
```

Zad. 2
Dany jest program w Prologu:

```
kocha(marcellus, mia).
kocha(vincent, mia).
zazdrosny(X,Y) :- kocha(X,Z),kocha(Y,Z).
```

Narysuj drzewko przeszukiwania dla zapytania: zazdrosny(X,Y). Jakie wyniki da zapytanie: zazdrosny(X,mia)?



zazdrosny(X,mia) -> false

<u>Zad. 3</u>

Napisać predykat, który pobiera dynamicznie od użytkownika imię żeńskie, potem sprawdza czy to imię jest w bazie. Jak nie ma to dodaje i potem wyświetla zawartość bazy. Dopisać polecenie, jakie trzeba dodać, żeby program działał dynamicznie.

```
:-dynamic(imie/1).
imie(zbyszek).
start:-
        write("Podaj Imie: "),nl,
        read(X),
        baza(X).
baza(X):-
        imie(X),write("Imie juz jest w bazie"),nl.
baza(X):-
        \+imie(X),assert(imie(X)),
        nl,write('Imie zostalo dodane do bazy!'),nl,
        listing(imie).
```

Druga wersja:

```
go :- read(Imie),
```

```
    \+imie(Imie),
    assertz(imie(Imie)),
    listing(imie).
```

Zad. 4

Dwie listy, które są posortowane. Napisać predykat *merge*, który połączy obie listy tak, że zachowają one swój porządek.

```
merge([], L, L).
merge(L, [], L).
merge([H1|R1], [H2|R2], [H1|R3]) :-
    H1 @< H2,
    merge(R1, [H2|R2], R3).
merge([H1|R1], [H2|R2], [H2|R3]) :-
    H1 @>= H2,
    merge([H1|R1], R2, R3).
```

Zad. 5

Dany jest program w Prologu, który powinien wyświetlić listę imion i napis: 'Oni są fajni':

```
student(szymon, agh).
student(krzysiek, agh).
student(weronika, agh).
student(kasia, agh).
student(szymon, agh).
fajni_studenci :- student(X, agh), write(X), write(' nie jest fajny'),!.
fajni_studenci :- write('oni są fajni').
```

Powyższy program nie działa tak jak w założeniu. Popraw jego działanie wstawiając w odpowiednie miejsce klauzulę 'fail'.

```
fajni :- student(X,agh),write(X),fail,write('nie jest fajny'),!.
fajni :- write('oni sa fajni').
```

Zad. 6

Napisać operator o priorytecie 80, który będzie robił tak, że będzie działało *sokrates jest czlowiek* zamiast *jest(czlowiek,sokrates)*

```
:- op(80, xfy, jest). %wersje (xfx, xfy) różnią się priorytetami po kazdej ze stron operatora. Dopóki nie zamierzamy robić wyrażeń zagnieżdżonych (tak jak na przykład 1+((2+1)+4) i tym podobnych) to nie ma znaczenia której wersji użyjemy
```

2015/16

<u>Zad. 1</u>
Dany jest fragment bazy faktów:

ksiazka(16, 'Flatlandia', autor('Edvin Abbot', 1838-1926), wydanie('Sell & Co', 1884)).
ksiazka(28, 'R. U. R.', autor('Karel Capek', 1890-1938), wydanie('Palyers Press', 2002)).
ksiazka(34, 'Kobieta z wydm', autor('Kobo Abe', 1924-1993), wydanie('Wydawnictwo Znak', 2007)).
ksiazka(36, 'Zamek', autor('Frans Kafka', 1883-1924), wydanie('Zielona Sowa', 2001)).
ksiazka(37, 'Gargantua i Pantagruel', autor('Francois Rabelais', 1494-1553), wydanie('Siedmiorόg', 2004)).

Należy napisać klauzule, które znajdują w bazie:

1. **pary wydawnictw, które wydały książkę o tym samym tytule (klauzula dwuargumentowa)**
2. **wydawnictwa, które wydawały książki danego autora po jego śmierci (klauzula dwuargumentowa)**
3. **listę wszystkich wydawnictw, które wydały przynajmniej jedną książkę napisaną przez autora urodzonego w XIX wieku (należy zastosować predykaty agregacyjne).**

```
tenSamTytul(Wyd1,Wyd2) :-
        ksiazka(_,Tytul,_,wydanie(Wyd1,_)),
        ksiazka(_,Tytul,_,wydanie(Wyd2,_)),
        Wyd1 \= Wyd2.

wydaniePosmiertne(Autor, Wydawnictwo) :-
        ksiazka(_,_,autor(Autor, _-RokSmierci),wydanie(Wydawnictwo,RokWydania)),
        RokSmierci @< RokWydania.

wydanieXIX(Wydawnictwo) :-
        ksiazka(_,_,autor(_, RokUrodzenia-_),wydanie(Wydawnictwo,_)),
        RokUrodzenia @< 1901, RokUrodzenia @> 1800.
setof(X, wydanieXIX(X), L).
```

<u>Zad. 3</u>
Napisz predykat *poznaj_slowo*, który:

1. **Poprosi użytkownika o podanie trudnego słowa**
2. **Pobierze od niego odpowiedź**
3. **Sprawdzi w bazie wiedzy czy takie słowo już w niej jest**
    a. **Jeżeli tak, to wypisze: “Dziękuję, ale to już znam.”**
    b. **Jeżeli nie ma, to wypisze komunikat: “A co to znaczy?”, pobierze odpowiedź, po czym zapisze w bazie wiedzy fakt**

definicja(NoweSlowo,Definicja) **korzystając z odpowiedzi udzielonej przez użytkownika.**

```
:- dynamic(definicja/2).
definicja(prolog, wspanialosc).
poznaj_slowo:-
        write("Podaj trudne slowo: "),nl,
        read(X),
        baza(X).
baza(X):-
        definicja(X,_),
        write("Dziekuje, ale to juz znam"),nl.
baza(X):-
        \+definicja(X,_),
        write('A co to znaczy?'),nl,
        read(Y),
        assert(definicja(X,Y)).
```

Zad. 4

Dana jest lista list. Napisz predykat, który pobierze 3. element z drugiej podlisty i 2. z trzeciej.

```
predykat([_,[_,_,X|_],[_,Y|_]|_],X,Y).
```

Zad. 5

Dany jest program w Prologu, który powinien sumować liczby umieszczone w liście.

```
nalezy(H, [H|_]).
nalezy(X,[_|Lista]) :- nalezy(X,Lista).
sumuj([X],X).
sumuj([H|Lista],Wynik) :-
        nalezy(X,[H|Lista]),sumuj(Lista,WynikNizej),Wynik is WynikNizej+X.
```

Program nie działa jednak tak jak powinien, ponieważ wyświetla więcej niż jeden wynik, podczas gdy poprawną odpowiedzią jest tylko pierwsza zwracana przez program. W jaki sposób używając operatora odcięcia w predykacie sumuj można poprawić działanie programu?

```
nalezy(H,[H|_]).
nalezy(X,[_|Lista]) :- nalezy(X,Lista).
sumuj([X],X).
sumuj([H|Lista],Wynik):-
        nalezy(X,[H|Lista]),sumuj(Lista,WynikNizej),Wynik is WynikNizej+X,!.
```

Zad. 6

Proszę zdefiniować operator *ma* o priorytecie 90 pozwalający na zapis klauzul w postaci *ala ma kota.* zamiast *ma(ala,kota).* Podaj przykład użycia. Jak zamienić term w listę?

```
:- op(90,xfy,ma).
ala ma kota.

<jakis term> =.. Lista.
```

Inne przydatne:

Wypisanie k-tego elementu listy:

```
wypisz([X|_],1) :-
        write(X).
wypisz([_|R],K) :-
        N is K-1,
        wypisz(R, N).
```

Usunięcie k-tego elementu listy:

```
usun([_|R],1,R).
usun([X|R],K,[X|L]) :-
        N is K-1,
        usun(R,N,L).
```

2015/16 poprawa

Zad. 1

Dany jest fragment bazy faktów:

ksiazka(16, 'Flatlandia', autor('Edvin Abbot', 1838-1926), wydanie('Sell & Co', 1884)).
ksiazka(28, 'R. U. R.', autor('Karel Capek', 1890-1938), wydanie('Palyers Press', 2002)).
ksiazka(34, 'Kobieta z wydm', autor('Kobo Abe', 1924-1993), wydanie('Wydawnictwo Znak', 2007)).
ksiazka(36, 'Zamek', autor('Frans Kafka', 1883-1924), wydanie('Zielona Sowa', 2001)).
ksiazka(37, 'Gargantua i Pantagruel', autor('Francois Rabelais', 1494-1553), wydanie('Siedmiorόg', 2004)).

Należy napisać klauzule, które znajdują w bazie:

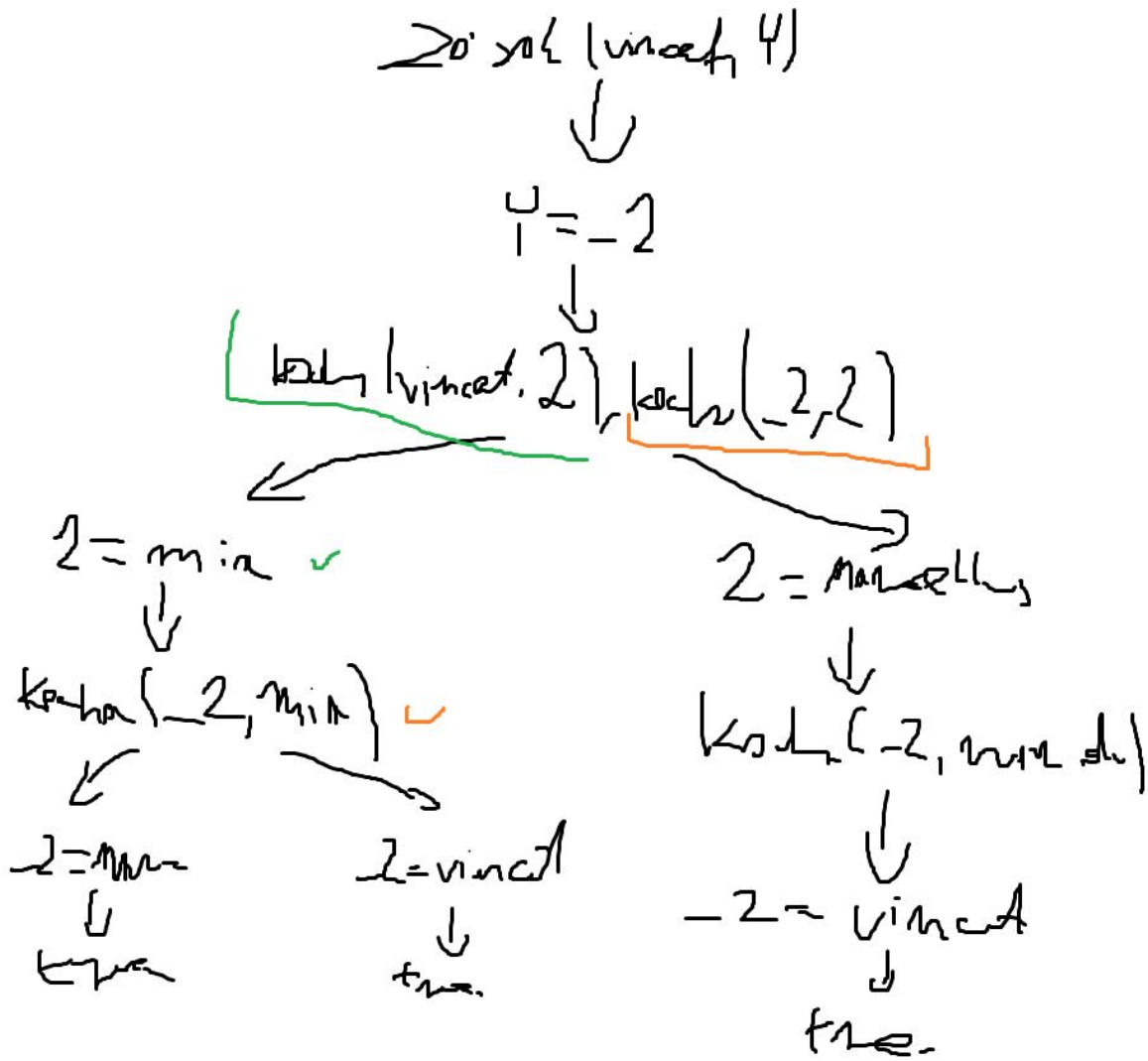
1. **pary wydawnictw, u których pisał pewien wspólny autor**
2. **książki wydane za życia danego autora**
3. **listę wszystkich wydawnictw, które wydały przynajmniej jedną książkę przed rokiem 1950 (należy zastosować predykaty agregacyjne).**

```
paryWydawnictw(X) :-
   ksiazka(_,_,autor(X,_-_),wydanie(W,_)),
   ksiazka(_, _, autor(X, _-_), wydanie(W,_)).

ksiazkiZaZycia(X) :-
   ksiazka(_,_,autor(X,_-Y),wydanie(_,Z)),
   Z < Y.

ksiazkiPrzed1950(Wyd) :-
   ksiazka(_,_,autor(_,_-_),wydanie(Wyd,Z)),
   Z < 1950.
setof(X, ksiazkiPrzed1950(X), L).
```

<u>Zad. 2</u>
Dany jest program w Prologu:

```
kocha(marcellus, mia).
kocha(vincent, mia).
kocha(vincent,marcellus).
zazdrosny(X,Y) :- kocha(X,Z),kocha(Y,Z).
```

Narysuj drzewko przeszukiwania dla zapytania: zazdrosny(vincent,Y). Jakie wyniki da zapytanie: zazdrosny(X,mia)?



Zad. 3

Proszę zdefiniować operator *ma* o priorytecie 90 pozwalający na zapis klauzul w postaci *ala ma kota.* zamiast *ma(ala,kota).* Podaj przykład użycia. Co gwarantuje, że wyrażenie *ala ma kota ma pchly.* będzie niedozwolone Jak zamienić term w listę?

:- op(90,xfx,ma).
ala ma kota.

<jakis term> =.. Lista.

Gwarantem jest xfx.

Zad. 4

Napisz predykat merge/3, który wstawi element do uporządkowanej niemalejąco listy z zachowaniem porządku. Jeśli przekazana do predykatu lista nie jest uporządkowana niemalejąco, to wywołanie ma się kończyć niepowodzeniem.

```
merge([], El, [El]).

merge([X|T], El, [El,X|T]) :-
        El =< X.

merge([X|T], El, [X|T2]) :-
        El > X,
        merge(T, El, T2).
```

Zad. 5

Dany jest program w Prologu, który powinien wyświetlić listę imion i napis: ‘Oni są fajni’:

```
student(szymon, agh).
student(krzysiek, agh).
student(weronika, agh).
student(kasia, agh).
fajni_studenci :- student(X, agh), write(X), write(' nie jest fajny'),!.
fajni_studenci :- write('oni są fajni'), fail.
fajni_studenci :- write('Ha! Tylko żartowałem!'). fail.
fajni_studenci.
```

Powyższy program nie działa tak jak w założeniu. Popraw jego działanie wstawiając w odpowiednie miejsce klauzulę ‘fail’.

```
fajni :- student(X,agh),write(X),fail,write('nie jest fajny'),!.
fajni :- fail, write('Oni sa fajni'), fail.
```

Zad. 6

Napisz predykat pozyskaj_miasto, który:

1. **Poprosi użytkownika o podanie nazwy miasta.**
2. **Pobierze od niego odpowiedź.**
3. **Sprawdzi w bazie wiedzy, czy takie miasto już w niej jest. Jeżeli tak, to wypisze: “Dziękuję, ale to już znam.”. Jeżeli nie ma, to dopisze podane miasto do bazy wiedzy (np. miasto(NoweMiasto)), wyświetli tekst: “Dziękuję! Teraz znam: “ i wypisze wszystkie znane miasta.**
   Czym się różni abolish/1 od retractall/1? Podaj przykłady użycia.

//

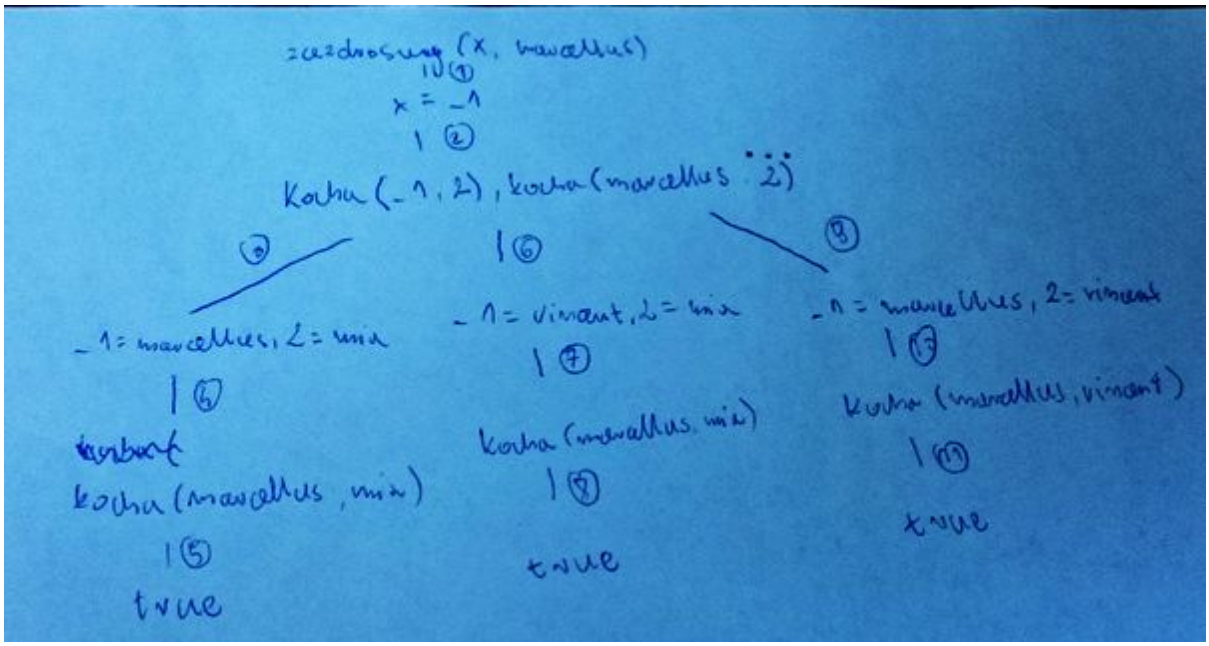
The retractall/1 standard built-in predicate can be used to remove all clauses for a dynamic predicate but the predicate will still be known by the runtime. The abolish/1 standard built-in predicate, in the other hand, not only removes all predicate clauses but also makes the predicate unknown to the runtime. If you try to call a dynamic predicate after removing all its clauses using retractall/1, the call simply fails. But if you abolish a dynamic predicate, calling it after will result in a predicate existence error.

Poprawa

Zad. 2
Dany jest program w Prologu:

kocha(marcellus, mia).
kocha(vincent, mia).
kocha(marcellus,vincent).
zazdrosny(X,Y) :- kocha(X,Z),kocha(Y,Z).

Narysuj drzewko przeszukiwania dla zapytania: zazdrosny(X,marcellus). Jakie wyniki da zapytanie: zazdrosny(marcellus,X)?



zazdrosny(marcellus,X):
X = marcellus,
X = vincent,
X = marcellus.

Zad. 4
generator(X) ma generować listę [1,2,3,...,X]

predykat podziel listę który działa tak:
dziel([1,2,3,4,5], [2], [1], [3,4,5])
dziel(1,2,3], [3], [1,2], [])

generator(0,[]):-!. %wykrzyknik to cut, zapobiega dalszym przeszukiwaniom i w konsekwencji przepełnieniu stosu

```
generator(X,L) :-
    N is X-1,
    generator(N,P),
    append(P,[X],L).    %append skleja dwie listy

dziel([],_,[],[]).  % bo listy pustej się nie podzieli inaczej
dziel([H|T],[H],[],T):-!. % jeśli pierwszy element listy jest delimiterem
dziel([H|T],[D],[H|L1],L2):-    % w innym wypadku zdejmij pierwszy element i dziel dalej
    dziel(T,[D],L1,L2).
```