1. Варіант № 28

1.1. Поясніть поняття «зелене відсікання». Наведіть приклад програми, в якій використовується зелене відсікання. Поясніть, чому в наведеному прикладі відсікання можна вважати зеленим.

Зелене відсікання — це відсікання, яке не змінює декларативного змісту предикату. Наприклад, розглянемо предикат `min/3`, який визначає мінімальне з двох чисел:

```
min(X, Y, X) :-
    X < Y,
    !.
min(X, Y, Y) :-
    X >= Y.
```

У першому правилі, якщо умова X < Y істинна, то і правило буде істинним, Пролог дасть правильну відповідь, відсікання зупинить відкат і не дасть Прологу перейти до другого правила — завершить пошук. Якщо видалити це відсікання, і умова X < Y буде істинною, перше правило буде істинним, але потім Пролог перейде до перевірки другого правила. Тим не менш, воно не буде істинним, тому Пролог також дасть єдину правильну відповідь. Отже, декларативний зміст предикату не змінився, тому таке відсікання є зеленим.

1.2. Поясніть, що таке анонімна змінна і для чого вона може бути використана в Пролог-програмі.

Анонімна змінна — це спеціальна змінна, яка позначається знаком підкреслення — «_». Щоразу, коли використовується анонімна змінна, вона пов'язується з новим значенням, тому показує, що її значення не важливе для обчислень. Анонімна змінна використовується для заповнення місця, коли синтаксис вимагає використати змінну, але її значення не потрібне. Наприклад, якщо для факту з декількох аргументів важливо знайти значення лише одного з них.

Наприклад, розглянемо факти, що описують машини і їх кольори:

```
car(honda01, red).
car(toyota_camry, blue).
car(mercedes_s500, black).
car(mercedes_gls450, black).
car(lexus_lfa, red).
```

Тоді щоб знайти, якщо потрібно знайти, яких кольорів є машини, але назви машин не важливі, використаємо анонімну змінну, щоб сформувати запит:

```
car(_, Color).
```

Таким чином отримаємо результат:

```
Color = red ;
Color = blue ;
Color = black ;
Color = black ;
Color = red.
```

1.3. Задача. Мовою Пролог опишіть предикат `subtract(L1, L2, L3)`, який виконує віднімання двох невпорядкованих списків. Тобто, породжує список, який містить елементи пергого вхідного списку, відсутні у другому вхідному списку.

```
% SWI Prolog
% member(X, L) перевіряє, чи є елемент X в списку L
member2(X, [X|_]).
member2(X, [_|Ys]) :-
    member2(X, Ys).

% subtract2(L1, L2, L3) видаляє зі списку L1 елементи списку L2 і
% повертає результат у списку L3. Інакше кажучи, він породжує такий
% список L3, в якому є лише ті елементи списку L1, яких немає у списку
% L2.
% Якщо L1 — порожній список, то яким би не був список L2, їх спільний
% елемент — порожній список (тобто жодного елемента)
subtract3([], _, []).
% Якщо елемент A є у списку B, не записувати його у список, і перейти
% до наступного елемента зі списку L1.
subtract3([A|As], B, R) :-
    member2(A, B),
    subtract3(As, B, R).
% Якщо ж елемента A немає у списку B, записати його у результуючий
```

```
% список, і перейти до наступного елемента зі списку L1.
subtract3([A|As], B, [A|Rs]) :-
    not(member2(A, B)),
    subtract3(As, B, Rs).
```