НАЦІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ

ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК

КАФЕДРА ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

МЕТОДИЧНІ ВКАЗІВКИ

щодо виконання лабораторних робіт

з курсу “Технологія програмування та створення програмних продуктів”

для студентів 4-го курсу

спеціальності “Інтелектуальні системи прийняття рішень”

Частина 1. Логічне програмування

КИЇВ — 2009

Методичні вказівки щодо виконання лабораторних робіт з курсу "Технологія програмування та створення програмних продуктів" для студентів 4-курсу з спеціальності "Інтелектуальні системи прийняття рішень". Частина 1. Логічне програмування / Укл. Баклан І.В., Степанкова Г.А. - К.: НАУ, 2009. - 44 с.

ЗМІСТ

ЗМІСТ.....3
Частина 1. МОВА ПРОГРАМУВАННЯ ПРОЛОГ.....4
Частина 2. ЛАБОРАТОРНІ РОБОТИ.....33

Частина 1. МОВА ПРОГРАМУВАННЯ ПРОЛОГ

Пролог (Prolog, програмування в логиці) - одна з найбільш широко використовуваних мов логічного програмування. Як і для інших декларативних мов, при роботі з нею ми описуємо ситуацію (правила й факти) і формулюємо мету (запит), дозволяючи інтерпретаторові Пролога знайти рішення задачі за нас.

Під інтерпретатором Прологу ми будемо розуміти **механізм вирішення задачі за допомогою мови** Пролог. Інакше кажучи, інтерпретатор мови Пролог - це виконавець Прологів-програм, тобто та "активна сила", що виконує програми, написані на Пролозі.

У кожної з мов програмування є своє коло задач, при вирішенні яких він використається з найбільшою ефективністю. Для Прологу це задачі, пов'язані з розробкою систем штучного інтелекту (різні експертні системи, програми-перекладачі, інтелектуальні ігри). Він використається для обробки природної мови й має потужні засоби, що дозволяють витягати інформацію з баз даних, причому методи пошуку, використовувані в ньому, принципово відрізняються від традиційних.

Пролог знайшов застосування й у ряді інших областей, наприклад, при вирішенні задач складання складних розкладів. При цьому він не є універсальною мовою програмування й не призначений, наприклад, для вирішення задач, пов'язаних із графікою або чисельними методами.

Існує велика кількість реалізацій мови Пролог, як комерційних, так і вільно розповсюджуваних. Ми будемо орієнтуватися на SWI-Prolog та GNU-Prolog, розроблений в університеті міста Амстердам. Можливостей даної реалізації цілком достатньо для первісного знайомства з основами логічного програмування.

SWI-Prolog та GNU-Prolog поширюється під ліцензією GPL, що забезпечує можливість його використання без порушень або комерційних інтересів. Ця версія мови Пролог доступний як користувачам ОС Linux, так і користувачам Windows.

1.1. Класична логіка та мова Пролог

Логічні мови, як можна побачити з їхньої назви, для мети передачі змісту програм використають засоби математичної логіки. Сама по собі логіка була винайдена як інструмент людської думки, що дозволяє впорядкувати знання й одержати з них відповідні висновки. Тому ідея використання принципів математичної логіки при складанні комп'ютерних програм здається досить природною.

Раніше ми вже познайомилися із частиною логіки, яка зветься численням висловлювань. Але обрахування висловлювань не дає можливості виразити багато фактів та міркувань, якими користуються в повсякденному житті. Наприклад, розглянемо класичне міркування:

```
Всі люди смертні (p);
Сократ - людина (q);
отже, (->)
Сократ смертний (r).
```

Це міркування вірне, але його неможливо довести в рамках теорії висловлювань. Ми можемо записати формулу (p&&q)->r, але довести її істинність уже не зможемо. Таким чином, логіка висловлень не дозволяє досить точно виразити розглянуте міркування. Це пов'язане з тим, що вона розглядає кожне висловлювання як неподільний об'єкт, у той час як багато висловлень залежать від якихось параметрів.

Вирахування предикатів є узагальненням вирахування висловлень, що дозволяють використати параметри (які також звуться аргументами або змінними) у висловленнях. У термінах теорії предикатів наше міркування можна записати так:

```
Для всіх x, якщо x є людиною,
те x є смертним;
Сократ є людиною;
(отже)
Сократ є смертним.
```

Вивчення вирахування предикатів не є нашою задачею, однак, для того, щоб застосовувати мову логічного програмування, не обов'язково знати логіку предикатів: вона вже вбудована в мову. Досить вивчити саму мову й звикнути до її виразних засобів.

Мова Пролог, найвідоміша із представників сімейства мов логічного програмування, вона зросла з робіт Алана Колмерауэра (A. Colmerauer) по обробці природної мови й незалежних робіт Роберта Ковальского (R. Kowalski) по використанню логіки у програмуванні. Дэвиду Уоррену (D. Warren) і його колегам з Эдинбургского університету вдалося здійснити досить ефективну реалізацію Прологу. Ім'я Уоррена ввійшло в історію логічного програмування. У його честь названа базова техніка реалізації Прологу, що одержала назву **абстрактної машини Уоррена**.

Програма мовою Пролог являє собою набір фактів й (можливо) правил. Якщо програма містить тільки факти, то її називають **базою даних**. Якщо вона містить ще й правила, то часто використають термін **база знань**.

Для запуску Прологу, наберіть у командному рядку pl і натисніть **Enter**. На екрані з'явиться запрошення для введення запитів:

```
?-
```

Запит (питання) вводиться після запрошення й обов'язково закінчується крапкою, наприклад,

```
?- 5+4<3.
No
```

Пролог аналізує запит і видає відповідь **Yes** (Так) у випадку істинності твердження й **No** (Ні) у противному випадку або коли відповідь не може бути знайдена.

Зберігають програми мовою Пролог у текстових файлах, що найчастіше мають розширення .pl, наприклад, example1.pl. Для того щоб Пролог міг оперувати інформацією, що розміщена у файлі, він повинен ознайомиться з його вмістом (проконсультуватися з ним). Це можна зробити декількома способами. При використанні першого варіанта у квадратних дужках записується ім'я файлу (без pl), наприклад,

```
?- [example1].
```

У випадку вдалого завершення цієї операції буде видане повідомлення, аналогічне наступному:

```
% example1 compiled 0.00 sec, 612 bytes
Yes
```

У противному випадку буде виданий список помилок (ERROR) і/або попереджень (Warning).

Другий спосіб базується на виклику вбудованого предиката consult, якому як аргумент передається ім'я файлу (також без розширення), наприклад:

```
?- consult(example1).
```

Розширення pl часто використається для файлів, що містять програми мовою програмування Perl, тому можна зустріти й інші розширення для файлів із програмами на Пролозі. Для завантаження файлів з розширеннями, відмінними від pl, все ім'я файлу варто обов'язково вкласти в апострофи:

```
?- consult('example2.prolog').
?- ['example2.prolog'].
```

Обидві ці команди додають факти й правила із зазначеного файлу в базу даних Прологу. Можна завантажувати кілька файлів одночасно. У цьому випадку вони додаються через кому, наприклад,

```
?- [example1, 'example2.prolog'].
```

Важливо пам'ятати, що всі запити повинні закінчуватися крапкою. Якщо

ви забудете її поставити, то Пролог виведе символ '|' і буде очікувати подальшого введення. У цьому випадку треба ввести крапку й нажати клавішу Enter:

```
?- [example1]
| .
Yes
```

1.2. Терми та об'єкти

Програма мовою Пролог зазвичай описує якусь дійсність. Об'єкти (елементи) описуваного світу представляються за допомогою термів. **Терм** інтуїтивно означає об'єкт. Існує 4 види термів: **атоми**, **числа**, **змінні** й **складні терми**. Атоми й числа іноді групують разом і називають найпростішими термами.

Атом - це окремий об'єкт, що вважається елементарним. У Пролозі атом представляється послідовністю літер нижнього й верхнього регістру, цифр та символів підкреслення '_', яка починається з малої літери. Крім того, будь-який набір припустимих символів, вкладений в апострофи, також є атомом. Нарешті, комбінації спеціальних символів + - * = < > : & також є атомами (слід зазначити, що набір цих символів може відрізнятися в різних версіях Прологу).

Приклад

Представлені далі послідовності є коректними атомами:

```
b, abcXYZ, x_123, efg_hij, оля, слюсар,
'Це також атом Прологу',
+, ::, <---->, ***
```

Числа в Пролозі бувають цілими (Integer) і дробовими (Float).

Синтаксис цілих чисел простий, як це видно з наступних прикладів: 1, 1313, 0, -97. Не всі цілі числа можуть бути представлені в машині, їхній діапазон обмежений інтервалом між деякими мінімальним і максимальним значеннями, які визначаються певною реалізацією Прологу. SWI-Prolog допускає використання цілих чисел у діапазоні від -2147483648 (-231) до 2147483647 (231-1).

Синтаксис дробових чисел також залежить від конкретної реалізації. Ми будемо дотримуватися простих правил, зрозумілих з наступних прикладів: 3.14, -0.0035, 100.2. При звичайному програмуванні на Пролозі дробові числа використаються рідко. Причина цього в тім, що Пролог - мова, призначена в першу чергу для обробки символьної, а не числової інформації. При символьній обробці часто використовуються цілі числа, потреба в дробових числах

невелика. Скрізь, де можна, Пролог намагається привести число до цілого виду.

Змінними в Пролозі є рядки символів, цифр і символу підкреслення, що починаються із великої букви або символу підкреслення:

```
X, _4711, X_1_2, Результат, _x23, Об'єкт2, _
```

Останній приклад (єдиний символ підкреслення) є особливим випадком - анонімної змінної (змінної без імені). Анонімна змінна застосовується, коли її значення не використається в програмі. Можливе кількаразове вживання безіменної змінної в одному вираженні застосовується для того, щоб підкреслити наявність змінних при відсутності їхньої специфічної значимості.

Складні терми (функції) складаються з імені функції (нечислового атома) і списку аргументів (термів Прологу, тобто атомів, чисел, змінних або інших складових термів), що взяті у круглі дужки й розділені комами. Групи складних термів використовують для складання фраз Прологу. Не можна розміщувати символ пробілу між функтором (ім'ям функції) і відкриваючою круглою дужкою. В інших позиціях, однак, пробіли можуть бути корисні для легшого читання програм. Нижче наведено два складні терми:

```
итого(клієнт(X,23,_), 71)
'Що трапилося?'(нічого)
```

При завданні імен термів переважніше використати мнемонічні імена, тому що терм а(ж), наприклад, набагато менш інформативний, ніж терм автор(жуль_верн).

Ще однією важливою структурою даних у Пролозі є список. Ми познайомимося з ним пізніше. Зараз відзначимо тільки один з видів списків - список символів. Такі списки можуть бути представлені у вигляді **рядків**, наприклад, перший аргумент складного терму вік("Борис",10) - рядок. При записі рядки беруться у лапки.

1.3. Факти

Програмувати на Пролозі - означає описувати якийсь світ. Програма на цій мові складається із множини **фраз**, що задають взаємозв'язок між термами. Кожен терм позначає ту або іншу сутність, що належить світу. Один зі способів опису - це завдання **фактів**.

Факт - це твердження про те, що існує деяке конкретне відношення. Він є безумовно вірним. У розмовній мові під фактом розуміється вислів типу "Сьогодні сонячно" або "Васі 10 років". На Пролозі це записується у вигляді

```
'Сьогодні сонячно'.
'Васі 10 років'.
```

Якщо ви збережете ці факти у файлі й потім завантажите його, то можна

задавати питання інтерпретаторові Прологу (нагадаємо, що запит вводиться після запрошення Прологу, що у більшості версій має вигляд ?-) , наприклад,

```
?- 'Сьогодні сонячно'.
Yes

?- 'Васі 10 років'.
Yes

?- 'Сьогодні сонячно', 'Васі 10 років'.
Yes
```

Кома між фактами в останньому запиті означає операцію логічного **І** (кон'юнкцію).

Така форма запису відповідає логіці висловлень, можливості якої, як уже говорилося, досить обмежені. Ми не можемо задати, наприклад, питання про те, скільки років Васі. Набагато зручніше використати **параметризовані** факти, роботу з якими підтримує логіка предикатів. На Пролозі факт може бути записаний у вигляді предиката, аргументи якого є символьними або числовими константами.

У загальному випадку **предикат** - це логічна функція від одного або декількох аргументів, тобто функція, що діє в множині з двох значень: істина та неправда. Предикат Прологу записується у вигляді складного терму:

```
ім'я_предикату(аргументи).
```

Аргументи перераховуються через кому та являють собою якісь об'єкти або властивості об'єктів, а ім'я предиката позначає зв'язок або відношення між аргументами. Предикат однозначно визначається парою: ім'я та кількість аргументів. Два предикати з однаковим ім'ям, але різною кількістю аргументів, вважаються різними. Кількість параметрів предиката називається його **арністтю** (arity). При описі предиката арність вказують після його імені, розділяючи їх символом '/' (слеш).

Як правило, імена предикатів й аргументів записуються в називному відмінку. Пробіли в них не допускаються, тому як роздільник в символьних константах використається символ підкреслення.

Приклад

Факт "Микола працює слюсарем" на Пролозі запишеться у такий спосіб:

```
професія(микола, слюсар).
```

Тут предикат **професія** має два аргументи: перший означає ім'я людини, а другий - професію. Факт "Борису 10 років" можна представити у вигляді:

```
вік("Борис", 10).
```

Порядок аргументів предиката зв'язаний зі змістом факту й тому не змінюємо. При записі фактів треба пам'ятати, що:

```
ім'я факту починається з малої літери;
запис кожного факту закінчується крапкою.
```

У наведених вище прикладах **професія** та **вік** - предикати (складні терми), **микола** та **слюсар** - атоми, **10** - число, **"Борис"** - рядок. Докладніше про види термів Прологу розповідається в наступному розділі.

База даних на Пролозі - це сукупність фактів. У процесі роботи в базу даних можна додавати нові факти, видаляти або змінювати старі.

Приклад

Складемо базу даних з наступних фактів:

"слон більше, ніж кінь",
"кінь більше, ніж віслюк",
"віслюк більше, ніж собака",
"віслюк більше, ніж мавпа":

```
більше(слон, кінь).
більше(кінь, віслюк).
більше(віслюк, собака).
більше(віслюк, мавпа).
```

Ми використали предикат більше, що має два параметри.

Збережемо цю базу даних у текстовому файлі та потім познайомимо Пролог з нею. Тепер можна формулювати запити до інтерпретатора Прологу:

```
?- більше(слон, кінь).
Yes

?- більше(кінь, слон).
No
```

1.4. Запити до бази даних

Запит - це послідовність предикатів, які розділені комами та завершені крапкою. Природною мовою кома відповідає союзу "і", а мовою математичної логіки позначає кон’юнкцію. За допомогою запитів можна "запитувати" базу даних про те, які твердження є істиними. Предикат запиту зветься **метою**.

Прості питання, що не містять ніяких змінних, називають **та-ні-питаннями**. Вони допускають лише дві можливі відповіді: "Yes" означає наявність відповідного факту в базі даних (перший запит приклада, наведеного нижче), "No" - його відсутність (другий запит). У випадку відповіді "Yes" говорять, що **запит завершився успіхом, ціль досягнута**.

Приклад

```
?- більше(слон, кінь), більше(кінь,вислюк).
Yes


?- більше(слон, собака).
No
```

Використання **змінних** у запитах дозволяє задавати більш складні питання. Припустимо, наприклад, що ми хочемо визначити, які тварини більше вислюка. У наступному запиті змінна X позначає шукану відповідь:

```
?- більше(X, вислюк).
X = кінь
Yes
```

При обробці запиту змінна X прийняла значення "кінь". Переглядаючи базу даних, інтерпретатор виявив факт, що стверджує, що кінь більше вислюка, та запит був успішно виконаний.

Запити зі змінними можуть мати більше одного рішення. Першим завжди виводиться те з рішень, що знаходиться ближче до початку бази даних. Якщо нам досить тільки однієї відповіді, то можна нажати **Enter** і закінчити пошук. У випадку, якщо ми захочемо одержати чергову відповідь, потрібно нажати клавішу ; (крапка з комою), і Пролог почне пошук інших варіантів відповіді на запит. Повідомлення "No" говорить про відсутність чергового рішення.

Приклад

```
?- більше(вислюк, X).
X = собака;
X = мавпа;
No


?- більше(X,Y).
X = слон
Y = кінь;

X = кінь
Y = вислюк;

X = вислюк
Y = собака;

X = вислюк
Y = мавпа;
No
```

1.5. Уніфікація

Уніфікація є основним механізмом обробки запитів у логічному програмуванні. Після того як користувач надсилає запит інтерпретаторові, цей запит **активізується**. Інтерпретатор приступає до аналізу фраз бази даних у пошуках першої фрази, заголовок якої буде уніфікуватися із запитом. Для того, щоб запит уніфікувався із заголовком фрази, необхідний збіг у них імені предиката, кількості аргументів та уніфікації кожного з них.

Уніфікація термів, якими є аргументи, описується правилами, які приведені нижче. У прикладах використається предикат «=», що намагається уніфікувати свої аргументи.

1. **Змінна уніфікується з атомом або складним термом**. У результаті цього змінна стає **конкретизованої**, тобто приймає значення даного атома або терму.

```
?- X=микола.
X=микола
Yes
```

2. **Змінна уніфікується зі змінною**, при цьому вони обидві стають як би однією й тією ж змінною.

```
?- X=Y.
X = _G161
Y = _G161
Yes
```

3. **Анонімна змінна уніфікується з будь-яким термом**.

```
?- автор(пушкін)=_.
Yes
```

4. **Атом уніфікується з атомом, якщо вони ідентичні**.

```
?- микола=микола.
Yes
```

5. **Складний терм уніфікується з іншим складним термом, якщо їхні імена й кількість аргументів збігаються, а аргументи піддаються уніфікації**.

```
?- батько(борис)=батько(X).
X = борис
Yes
?- дідусь(борис, Y)=батько(X).
No
```

Приклад

Терми більше(X, собака) і більше(віслюк, собака) уніфікуються, тому що змінна X може бути конкретизована атомом віслюк:

```
?- більше(X,собака) = більше(віслюк,собака).
X = віслюк
Yes
```

Розглянутий у наступному прикладі запит не буде успішним, тому що змінна X не може бути конкретизована двома значеннями 1 й 2 одночасно.

```
?- p(X,2,2) = p(1,Y,X).
No
```

Якщо в цьому прикладі замість X ми використаємо анонімну змінну _, то уніфікація буде можлива, тому що при кожному використанні _ створюється нова змінна. Зміст анонімності в тім, що ми надаємо Прологу можливість генерації імені для даної змінної й нам не потрібні ні її ім'я, ні її значення. Змінна Y під час уніфікації конкретизується значенням 2:

```
?- p(_,2,2) = p(1,Y,_).
Y = 2
Yes
```

Приклад

У наступному запиті уніфікація можлива, хоча й немає специфічних змінних, які могли б бути зв'язані або уніфіковані (як у попередніх прикладах):

```
?- f(a,g(X,Y)) = f(X,Z), Z = g(W,h(x)).
X = a
Y = h(x)
Z = g(a, h(x))
W = a
Yes
```

1.6. Правила

Крім фактів програми мовою Пролог можуть містити **правила**, що дозволяють одержувати додаткові знання про той світ, що описує програма. **Правило** задає новий предикат через визначені раніше предикати.

Правило складається з голови (предиката) та тіла (послідовності предикатів, розділених комами). Голова й тіло розділені знайомий :- і, подібно кожній фразі Прологу, правило повинне закінчуватися крапкою. Кома в тілі правила означає кон’юнкцію (&&, логічне **і**).

Знак :- є схематичним записом стрілки (<-) і показує, що із правої частини випливає ліва. Цей знак читається як "**якщо**". Інтуїтивний зміст правила полягає в тому, що ціль, яка є головою, буде істинною, якщо Пролог зможе

показати, що всі вирази у тілі правила є істинними.

Приклад

Правило, що визначає відношення «дитина» через відношення «батько», запишеться у такий спосіб:

```
дитина(X, Y) :- батько(Y, X).
```

Це означає, що якщо людина Y є для людини X батьком, то X є дитиною Y. Тут X й Y - змінні. Нагадаємо, що запис «дитина» показує, що предикат дитина є функцією від двох аргументів.

Приклад

Визначимо відношення «мати» через відносини «батько» та «жінка» у такий спосіб: матір'ю X для людини Y є його батько жіночого роду.

```
мати(X, Y) :- батько(X, Y), жінка(X).
```

Предикати відрізняються друг від друга не тільки ім'ям, але й кількістю аргументів. Можна, наприклад, визначити відношення «мати» у такий спосіб:

```
мати(X) :- батько(X, _), жінка(X).
```

Через те, що нам у даному предикаті не важливо, чиїм батьком є дана жінка, то ми використали анонімну змінну.

```
?- мати(X, Y).
X=ганна
Y=юлія
Yes

?- мати(X).
X=ганна
Yes
```

Приклад

Визначимо відношення «дідусь»:

```
дідусь(X, Y) :- батько(X, Z), батько(Z, Y).
дідусь(X, Y) :- батько(X, Z), мати(Z, Y).
```

Ці правила стверджують, що дідусем X для людини Y є батько людини Z, що у свою чергу є батьком або матір'ю людини Y.

1.7. Рекурсивні процедури

Рекурсія в більшості мов програмування - це такий спосіб організації обробки даних, при якому програма (процедура) викликає сама себе

безпосередньо, або за допомогою іншої програми (процедури).

Гравюра голандського художника Моріса Ешера "Руки, що малюють" (рис.1.) - одна з найкращих ілюстрацій поняття рекурсії. Усім відомий віршик про попа і його собаку демонструє нам нескінченність рекурсивних викликів. Використовуючи рекурсію як прийом програмування ми повинні бути впевнені, що рекурсивна процедура буде завершена.

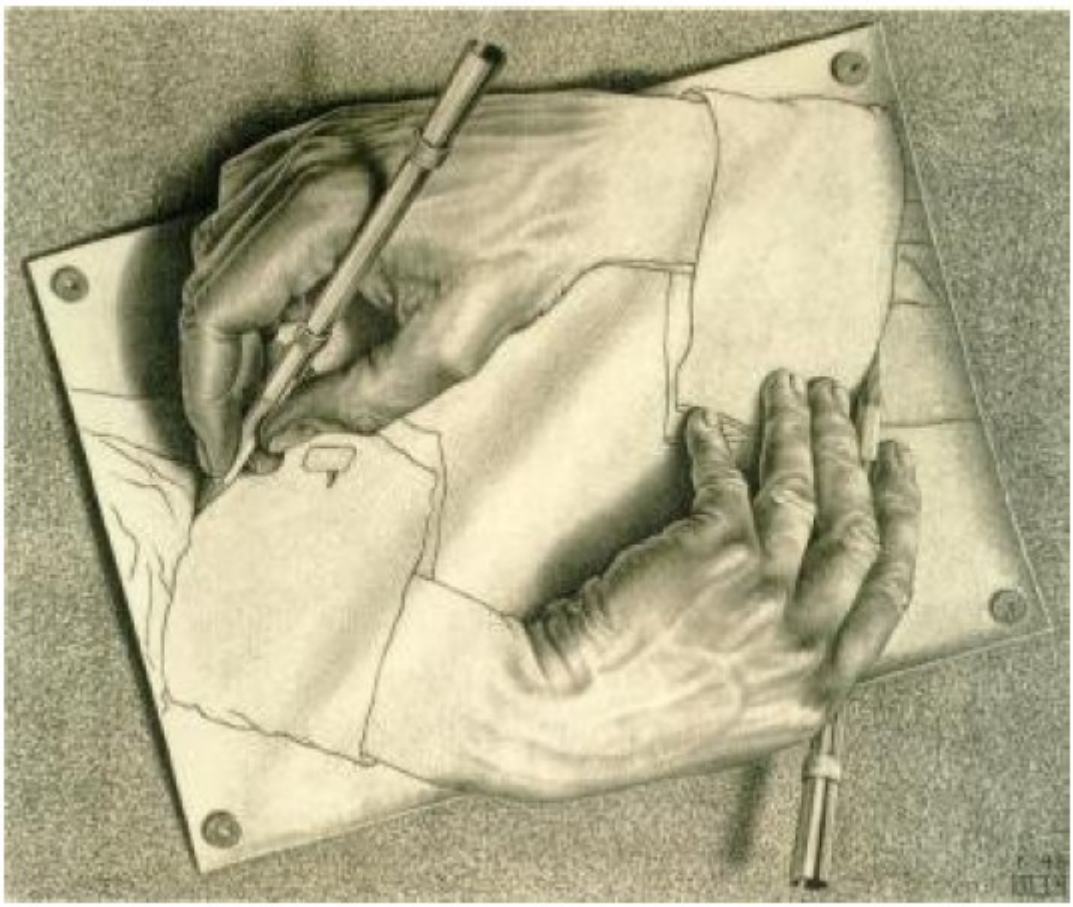


Рис. 1. - Гравюра голандського художника Моріса Ешера "Руки, що малюють"

У Пролозі рекурсія зустрічається, коли предикат містить мету, що посилається на саму себе. У рекурсивному правилі більш складні вхідні аргументи повинні виражатися через менш складні.

На прикладі вже наявної в нас бази даних пояснимо переваги використання рекурсії й особливості рекурсивних правил. Нехай є наступні факти:

```
більше(слон, кінь).
більше(кінь, вислюк).
більше(вислюк, собака).
більше(вислюк, мавпа).
```

Виконаємо запит до бази даних

```
?- більше(віслюк, собака).
Yes
```

Ціль **більше(віслюк, собака)** була досягнута тому, що цей факт був повідомлений Прологу при завантаженні бази. Тепер перевіримо, чи більше мавпа слона?

Ні, не більше. Ми одержали таку відповідь, яку й очікували: відповідний запит, а саме **більше(мавпа, слон)** не підтвердився. Але, що трапиться, якщо ми поставимо запитання по-іншому?

```
?- більше(слон, мавпа).
No
```

Таким чином, слони не більше, ніж мавпи. Отриманий результат зовсім не узгоджується з нашими уявленням про світ, але якщо подивитися на базу даних, то легко помітити, що в ній дійсно нічого не сказано про відносини між слонами й мавпами. Однак, ми знаємо, що слони більше, ніж кінь, який у свою чергу більший, ніж віслюк, який більший за мавпу, тому слони також повинні бути більші, ніж мавпи.

Правильна інтерпретація негативної відповіді, даної Прологом, така: інформації, яка повідомлена системі, недостатньо для доказу того, що слон більший за мавпу. Якщо ми захочемо одержати позитивну відповідь на запит виду **більше(слон, мавпа)**, то ми повинні забезпечити більш точний опис світу. Одним з можливих способів вирішення цієї проблеми є додавання відсутніх фактів, наприклад,

```
більше(слон, мавпа).
```

Для нашого маленького приклада це означає додавання ще 5 фактів. Однак набагато кращим рішенням буде додавання в програму нового відношення, що ми назвемо більше_2. Тварина X більша, ніж тварина Y, якщо це визначено як факт (перше правило) або існує тварина Z, для якої визначений факт, що тварина X більша, ніж тварина Z, і може бути показано, що тварина Z більша, ніж тварина Y (друге правило). На Пролозі це запишеться так:

```
більше_2(X, Y) :- більше(X, Y).
більше_2(X, Y) :- більше(X, Z), більше(Z, Y).
```

Якщо в ланцюжку беруть участь не три, а більше число об'єктів, то прийде додати нові правила:

```
більше_2(X, Y) :- більше(X, Z1), більше(Z1, Z2),
                  більше(Z2, Y).
більше_2(X, Y) :- більше(X, Z1), більше(Z1, Z2),
                  більше(Z2, Z3), більше(Z3, Y).
...
```

Ця програма довга й працювати буде далеко не завжди. Вона зможе

переглядати базу даних тільки до певної глибини, що задається максимальною кількістю підцілей у правилах.

Тому скористаємося більш коректним й елегантним формулюванням. Ключова ідея тут - **визначити відношення більше_2** за допомогою його самого. Тепер друге (і останнє!) правило виглядає так:

```
більше_2(X, Y) :- більше(X, Z), більше_2(Z, Y).
```

Таким чином, підсумкова програма буде мати вигляд

```
більше_2(X, Y) :- більше(X, Y).
більше_2(X, Y) :- більше(X, Z), більше_2(Z, Y).
```

Зверніть увагу на порядок підцілей у другому правилі: якщо їх поміняти місцями, то в більшості реалізацій мови Пролог виконання запиту до такої бази знань приведе до повідомлення про помилку, аналогічне наступному:

```
ERROR: Out of local stack
```

Якщо тепер у запиті використати предикат **більше_2** замість **більше**, те програма буде працювати так, як і передбачалося:

```
?- більше_2(слон, мавпа).
Yes
```

Інтерпретатор завжди переглядає базу даних **зверху вниз**. Тому він аналізує спочатку першу фразу процедури **більше_2** і намагається уніфікувати кожен аргумент запиту з відповідним аргументом цієї фрази. Це відбувається за допомогою **порівняння** запиту з початком правила **більше_2(X, Y)** (тобто з його головою). Після цього двом змінним привласнюються значення: **X = слон** та **Y = мавпа**.

Після конкретизації змінної деяким термом це значення "закріплюється" **за всіма випадками** використання цієї змінної в правилі. Після уніфікації запиту із заголовком фрази інтерпретатор переходить до обробки цілей, що знаходяться в тілі цієї фрази.

У цьому випадку Пролог не може знайти в базі даних факту **більше(слон, мавпа)** і переходить до розгляду другого правила. Воно говорить, що для того, щоб одержати відповідь на питання **більше_2(X,Y)** (з фіксованими значеннями змінних, тобто **більше_2(слон, мавпа)**), Пролог повинен відповісти на два **підпитання** **більше(X, Z)** і **більше_2(Z, Y)**, знову ж з відповідними значеннями змінних. Процес перегляду бази знань із самого початку повторюється доти, поки факти, що становлять ланцюжок між слон і мавпа, не будуть знайдені, а запит успішно оброблений.

Будь-яка рекурсивна процедура повинна включати принаймні по одній з нижче перерахованих компонентів.

1. **Нерекурсивну фразу**, що визначає правило, застосовуване в момент

припинення рекурсії.

2. **Рекурсивне правило**, перша підціль якого виробляє нові значення аргументів, а друга - рекурсивна підціль- використовує ці значення.

1.8. Бази знань

Як ми вже відзначали, програма мовою Пролог, що містить факти й правила, становить базу знань. При розробці програм на Пролозі часто використовують вбудовані предикати, тобто предикати, обумовлені автоматично при ініціалізації інтерпретатора Прологу.

Вбудовані предикати використовуються так само, як й обумовлені користувачем предикати. Єдине обмеження - вбудований предикат не може бути головою правила або з'являтися у факті.

Одним з найчастіше використовуваних вбудованих предикатів є предикат **not** (заперечення). Цей предикат істинний, якщо його аргумент помилковий, і навпаки. Можна використати й іншу форму запису даного предиката \+.

Приклад

Якщо ми визначимо правило

```
неправда(X) :- not(X).
неправда1(X) :- \+(X).
```

то наступні запити будуть еквівалентні:

```
?- not(більше(собака, кінь)).
Yes;

?- неправда(більше(собака, кінь)).
Yes
```

Іншим часто використовуваним вбудованим предикатом є = (уніфікація): **=(X, Y)**. Цей предикат допускає більше зручну форму запису **X = Y**. Значення цього предиката істинно, якщо терми X й Y вдається уніфікувати.

На предикат **not** схожий вбудований предикат \=, що залежить від двох аргументів. Твердження **X \= Y** еквівалентно твердженню **not(X = Y)**.

Іноді буває корисно використати предикати, про які заздалегідь відомо, істинні вони або помилкові. Для цих цілей використовують предикати **true** й **fail**. Предикат **true** завжди істинний, у той час як **fail** завжди помилковий.

Вбудований предикат **read** дозволяє зчитувати терми із клавіатури. При цьому запрошення Прологу **?-** міняється на **|:**. Терм, що вводиться, повинен

обов'язково закінчуватися крапкою.

Приклад

```
?- read(Name), read(Age).
|: микола. 15.

Name = микола
Age = 15
Yes

?- read(X), більше_2(X,Y).
|: віслюк.

X = віслюк
Y = собака ;

X = віслюк
Y = мавпа ;
No
```

Якщо при обробці запитів Прологу ви побажаєте одержати більш докладний висновок, то для цих цілей можна використати предикат **write**. Аргументом цього предиката може бути будь-який припустимий терм Прологу. У випадку, коли аргументом є змінна, буде надруковане її значення.

Виконання предиката **nl** здійснює переніс рядка: наступний висновок почнеться з нового рядка. Предикат **tab** виводить кількість пробілів, визначену його аргументом.

Приклад

```
?- write('Hello World!').
Hello World!
Yes

?- write('Hello'), nl, tab(5), write('World!').
Hello
     World!
Yes

?- X = слон, write(X), nl.
слон

X = слон
Yes
```

В останньому прикладі спочатку змінна X уніфікується з атомом слон, а потім **значення змінної** X, тобто слон, виводиться на екран за допомогою предиката **write**. Після переходу на новий рядок Пролог видає звіт про уніфіковану змінну, тобто друкує **X = слон**.

Більшість Пролог-систем надає доступ до довідкової інформації при виклику предиката **help**. Застосований до терму (звичайно представляє ім'я вбудованого предиката) він виводить короткий опис цього терму.

Приклад

```
?- help(write).
write(+Term)
  Write Term to the current output, using brackets and
operators
  where appropriate. See feature/2 for contrillong   floating
point
  output format.

write(+Stream, +Term)
  Write Term to Stream.
Yes
```

І, наостанок, поговоримо про **коментарі**. Коментарі ніяк не впливають на виконання програми, але при їх правильному використанні вони являються досить істотною частиною вихідного тексту. Трохи вдало розташованих рядків з коментарями дуже допоможуть людині, яка читає програму. Пролог ігнорує довільне число рядків, укладене між символами /* й */. Усе, що перебуває між **%** і кінцем рядка, теж розглядається як коментар:

Приклад

```
/* Це
        коментар */

 % Це теж коментар
```

1.9. Рішення логічних задач

Інтерпретатор Прологу можна змусити вирішувати логічні задачі, що недоступно більшості процедурних мов.

Багато логічних задач пов'язані з розглядом декількох кінцевих множин з однаковою кількістю елементів, між якими встановлюється взаємо-однозначна відповідність. Мовою Пролог ці множини можна описувати як бази даних, а залежності між об'єктами встановлювати за допомогою правил.

Приклад

В автомобільних перегонах три перших місця зайняли Олексій, Петро й Микола. Яке місце зайняв кожний з них, якщо Петро зайняв не друге й не третє місце, а Микола - не третє?

I'мя	I місто	II місто	III місто
Олексій			
Петро		-	-
Микола			-

Традиційним способом задача вирішується заповненням таблиці. За умовою задачі Петро зайняв не друге й не третє місце, а Микола - не третє. Це дозволяє поставити символ '-' у відповідних клітках. Між множиною імен учасників перегонів й множиною місць повинне бути встановлене взаємо-однозначна відповідність. Тому визначаємо зайняте місце спочатку в Петра, потім у Миколи й, нарешті, в Олексія. У відповідних клітках проставляємо знак '+'. У кожному рядку й кожному стовпці повинен бути тільки один такий знак.

I'мя	I місто	II місто	III місто
Олексій	-	-	+
Петро	+	-	-
Микола	-	+	-

З останньої таблиці випливає, що Олексій посів третє місце, Петро - перше, Микола - друге.

Мовою Пролог структура програми буде наступною: спочатку перераховуються дані - імена й номер зайнятого місця, а потім записуються правила, що зв'язують ці дві множини.

```
/* База даних імен */
ім'я(олексій).
ім'я(петро).
ім'я(микола).

/* База даних призових місць  */
місце(перше).
місце(друге).
місце(третє).

/* Встановлюємо взаємо-однозначну відповідність
між базами даних, X - елемент із бази даних імен,
```

```
Y - елемент із бази даних зайнятих місць */

/* Петро зайняв не друге й не третє місце */
відповідність(X, Y) :- ім'я(X),  X=петро,
         місце(Y), not(Y=друге), not(Y=третє).

/* Микола посів не третє місце */
відповідність(X, Y) :- ім'я(X), X=микола,
         місце(Y), not(Y=третє).

відповідність(X, Y) :- ім'я(X),  X=олексій, місце(Y).

/* У всіх хлопців різні місця */
рішення(X1,Y1,X2,Y2,X3,Y3) :-
         X1=петро, відповідність(X1,Y1),
         X2=микола, відповідність(X2,Y2),
         X3=олексій, відповідність(X3,Y3),
         Y1\=Y2, Y2\=Y3, Y1\=Y3.
```

Для одержання відповіді варто виконати запит

```
?- рішення(X1,Y1,X2,Y2,X3,Y3).
```

У відповідь Пролог видасть імена хлопців і зайняті ними місця. Перевірте, чи є інші варіанти відповідей.

Приклад

Вітя, Юра та Михайло сиділи на лавці. У якому порядку вони сиділи, якщо відомо, що Михайло сидів ліворуч від Юри, а Вітя ліворуч від Михайла.

В умові задачі перераховуються об'єкти одного типу, зв'язані між собою. Заповнимо базу даних:

```
/* Михайло сидів ліворуч від Юри  */
ліворуч(юра,михайло).

/* Вітя сидів ліворуч від Михайла */
ліворуч(михайло,вітя).
```

Правило для встановлення проходження об'єктів один за одним буде таким:

```
/* Об'єкти X, Y й Z утворять ряд,
якщо X ліворуч від Y й Y ліворуч від Z */

ряд(X, Y, Z) :- ліворуч(Y, X), ліворуч(Z, Y).
```

Кількість аргументів у голові даного правила дорівнює кількості об'єктів у

задачі. Запит **?- ряд(X, Y, Z)**. дасть нам рішення задачі.

1.10. Арифметичні вирази

У мові Пролог є ряд вбудованих функцій для обчислення арифметичних виразів, деякі з яких приведені в таблиці.

X + Y	Сума X та Y
X - Y	Різниця X та Y
X * Y	Добуток X та Y
X / Y	Ділення X на Y
X mod Y	Остача від ділення X на Y
X // Y	Ділення націло X на Y
X ** Y	Піднесення X у ступінь Y
- X	Зміна знака X
abs(X)	Абсолютна величина числа X
max(X,Y)	Больше з чисел X та Y
min(X,Y)	Менше з чисел X та Y
sqrt(X)	Квадратний корінь з X
random(Int)	Випадкове ціле число в діапазоні від 0 до Int
sin(X)	Синус X
cos(X)	Косинус X
tan(X)	Тангенс X
log(X)	Натуральний логарифм (ln) числа X
log10(X)	Десятковий логарифм (lg) числа X
float(X)	Дробове число, що відповідає цілому числу X
pi	3.14159 (наближене значення числа π)
e	2 .71828 (наближене значення числа e)

Ще раз відзначимо, що Пролог намагається сховати різницю між арифметикою цілих і дробових чисел скрізь, де це можна.

Для обчислення арифметичних виразів у Пролозі використається вбудований бінарний оператор **is**, що інтерпретує правий терм як арифметичний вираз, після чого уніфікує (якщо можливо) результат обчислення з лівим термом (звичайно зі змінною). Пріоритет виконання арифметичних операцій є традиційним. Круглі дужки використаються для зміни порядку обчислень. У наступних прикладах змінна X уніфікується зі значеннями арифметичних

виразів:

```
?- X is 2.5 + 2.5.
X = 5
Yes

?- X is 4/(2+1).
X =  1.33333
Yes

?- X is cos(3*pi).
X = -1
Yes

?- 1 is sin(pi/2).
Yes

?- 1.0 is sin(pi/2).
No
```

Пояснимо трохи несподівану відповідь Прологу в останньому запиті. Значення **sin(pi/2)** автоматично округляється предикатом **is** до цілого значення 1, що не вдається уніфікувати з дробовими числом 1.0. Предикат **float** змусить вважати значення **sin(pi/2)** дробовими числом:

```
?- 1.0 is float( sin(pi/2)).
Yes
```

Для порівняння арифметичних виразів використається ряд операторів. Ціль **X > Y** (більше) буде успішна, якщо вираз X буде відповідати більшому числу, ніж вираз Y.

Аналогічно використаються оператори **<** (менше), **=<** (менше або дорівнює), **>=** (більше або дорівнює), **=\=** (не дорівнює) і **=:=** (арифметично рівний). Розходження між операторами **=:=** й **=** дуже істотні. Перший оператор порівнює значення арифметичних виражень, тоді як останній намагається уніфікувати їх.

Приклад

```
?- 2 ** 3 =:= 3 + 5.
Yes

?- 2 ** 3 = 3 + 5.
No

?- 1.0 = float(sin(pi/2)).
No
```

```
?- 1.0 =:= sin(pi/2).
Yes
```

Помітьте, що ціль **X =:= Y** буде істинна, навіть якщо один з термів є ціле число, а іншої - рівне йому дробове.

Приклад

Порядок підцілей у запиті впливає на його результат:

```
?- X is 4+Y, Y=3.
ERROR: Arguments are not sufficiently instantiated

?- Y=3, X is 4+Y.
Y = 3
X = 7
Yes
```

У першому запиті повідомлення про помилку з'явилося тому, що перша підціль запиту (**X is 4+Y**) зазнала невдачі, тому що в момент її обробки неможливо обчислити вираз **4+Y**.

1.11. Приклади програм

Розглянемо деякі програми, що демонструють обробку числових даних. Відзначимо важливу особливість процедур, створюваних мовою Пролог: вони, у відмінності від вбудованих функцій, не можуть з'являтися в арифметичних виразах. Якщо потрібно, наприклад, змінної **R** привласнити значення, рівне помноженому на три більшому із двох виразів X й Y, то, використовуючи введену нижче процедуру **максимум**, це можна записати так:

```
максимум(X,Y,Z), R is 3*Z.

максимум(X,X,X).
максимум(X,Y,X):- X>Y.
максимум(X,Y,Y):- X<Y.

гіпотенуза(X,Y,Z):- number(X), number(Y), Z is sqrt(X**2 +
Y**2).

мін_гіп(A1,B1,A2,B2,Min):-
    гіпотенуза(A1,B1,C1),
    гіпотенуза(A2,B2,C2),
    Min is min(C1,C2).
```

```
сума(X,Y):- integer(X), X<10, Y is X.
сума(X,Y):- integer(X), X1 is X//10, сума(X1,Y1),
             Z is X mod 10, Y is Y1+Z.

друк_суми:-  write('Введіть число (не забудьте крапку
наприкінці): '),
    read(X), nl,
    write('Сума цифр введеного числа дорівнює '),
    сума(X,Y), write(Y), nl.

факт(1,1).
факт(N,R):- integer(N), N>1, N1 is N-1,
            факт(N1,R1), R is N*R1.

сума_списку([],0).
сума_списку([H|T],S):- сума_списку(T,S1), number(H), S is
S1+H.
```

Приклад

Написати процедуру, що обчислює максимум із двох чисел.

```
максимум(X,X,X).
максимум(X,Y,X):- X>Y.
максимум(X,Y,Y):- X<Y.
```

У предикаті **максимум** третій аргумент є максимумом з двох чисел – першого та другого його аргументів. Зміст кожного із правил даної процедури цілком очевидний. Подивимося на реакцію інтерпретатора Прологу на запити, що містять даний предикат.

```
?- максимум(20,50,X).
X = 50
Yes

?- максимум(100,50,X).
X = 100
Yes

?- максимум(X,50,100).
X = 100
Yes
```

Остання відповідь показує, що наш предикат дозволяє знаходити відповідь на питання типу: "Яке повинне бути число, щоб максимум із шуканого числа й числа 50 дорівнювався 100?".

Як ви думаєте, чому була отримана відповідь "No" на наступний запит?

```
?- максимум(X,50,40).
No
```

Приклад

Складіть процедуру **гіпотенуза**, що по двох катетах прямокутного трикутника обчислює його гіпотенузу.

Скористаємося теоремою Піфагора та вбудованою функцією **sqrt** для обчислення квадратного кореня:

```
гіпотенуза(X,Y,Z):- Z is sqrt(X**2 + Y**2).
```

Програма коректно обчислює гіпотенузу, але якщо ми спробуємо за її допомогою знайти один з катетів, то переконаємося, що процедура працює не цілком вірно. Щоб уникнути цього додамо перевірку того, що перші два аргументи предиката - позитивні числа, для чого використаємо вбудований предикат **number** і порівняння з нулем:

```
гіпотенуза(X,Y,Z):- number(X), X>0, number(Y), Y>0,
                     Z is sqrt(X**2 + Y**2).
?- гіпотенуза(3,4,X).
X = 5
Yes

?- гіпотенуза(3,'a',X).
No

?- гіпотенуза(3,X,5).
No
```

Приклад

Напишіть предикат, що по двох парах чисел - довжинам катетів прямокутних трикутників - визначає величину меншої з гіпотенуз.

Скористаймося процедурою **гіпотенуза**, яка розібрана вище, та вбудованою функцією **min**:

```
мін_гіп(A1,B1,A2,B2,Min):-
                  гіпотенуза(A1,B1,C1),
                  гіпотенуза(A2,B2,C2),
                  Min is min(C1,C2).
```

Запити до інтерпретатора Прологу можуть виглядати так:

```
?- мін_гіп(3,4,8,6,X).
X = 5
Yes

?- мін_гіп(3,4,Y,6,X).
```

```
No
```

Приклад

Факторіалом натурального числа n є добуток всіх цілих чисел від 1 до n включно. Для запису факторіала числа n використають позначення n!.

```
n!=n*(n-1)*(n-2)*...*2*1=n*(n-1)!
```

Наступна процедура обчислює факторіал числа. Зверніть увагу на використання **рекурсії** в даній процедурі:

```
факторіал(1,1).
факторіал(N,R):- integer(N), N>1, N1 is N-1,
                 факторіал(N1,R1), R is N*R1.
```

Перше правило (так званий термінальний випадок, тобто той момент виконання процедури, коли вона перестає викликати сама себе) говорить, що факторіал одиниці дорівнює одиниці. Друге правило є просто запис визначення факторіала: результат R виходить множенням числа N на факторіал числа, яке на одиницю менше. Воно буде спрацьовувати при всіх n>1 тому, що інтерпретатор Прологу переглядає базу даних зверху вниз і переходить до наступного правила або факту тільки в тому випадку, коли він не може виконати поточне правило.

Приклад

Напишіть програму мовою Пролог, що друкує суму всіх цифр введеного з клавіатури числа.

Для рішення даної задачі скористаємося двома предикатами. Предикат **сума** має своїм першим аргументом число, сума цифр якого є його другим аргументом. Другий предикат - **друк_суми** - запитує число, викликає предикат **сума** та друкує отриманий результат.

```
сума(X,Y):- integer(X), X<10, Y is X.
сума(X,Y):- integer(X), X1 is X//10, сума(X1,Y1),
            Z is X mod 10, Y is Y1+Z.

друк_суми:-  write('Введіть число (наприкінці крапка): '),
               read(X), nl, сума(X,Y),
               write('Сума цифр  числа '), write(X),
      write(' дорівнює '), write(Y), nl.
```

Правило **друк_суми** не має аргументів, дані вводяться з клавіатури й потім, за допомогою механізму уніфікації, передаються іншим підцілям даного правила.

Приклад

Напишіть програму мовою Пролог, що вводить з клавіатури два числа - координати точки на площині та визначає, чи попадає дана точка в коло одиничного радіуса із центром на початку координат.

```
inside(X,Y,попадає):- number(X), number(Y),
                      X**2+Y**2=<1.
inside(X,Y,не_попадає):-number(X), number(Y),
                      X**2+Y**2>1.

/* Ввести два числа та викликати предикат inside */

input:-write('Введіть x-координату: '),
       read(X), nl,
       write('Введіть y-координату: '),
       read(Y), nl,
       inside(X,Y,R), write(R).
```

1.12. Списки

Списки - одна з найчастіше вживаних структур у Пролозі. При записі список беруть у квадратні дужки, а елементи списку розділяють комами, наприклад,

```
[слон, кінь, мавпа, собака]
```

Це список із чотирьох атомів - слон, кінь, мавпа, собака.

Елементами списку можуть бути будь-які терми Прологу, тобто атоми, числа, змінні й складні терми, що дозволяє, зокрема, складати списки зі списків. Порожній список записується як [ ].

```
[слон, [ ], X, предок(X, том), [a,b,c], f(22)]
```

Перший елемент непорожнього списку називається **головою**, а інша частина списку зветься **хвіст**. У списку, що складається тільки з одного елемента, головою є цей єдиний елемент, а хвостом - порожній список. Позначення [H|T] використовується для визначення списку з головою H і хвостом T. Якщо символ | розміщений перед останнім термом списку, то це означає, що цей останній терм визначає інший список. Повний список вийде, якщо з'єднати цей підсписок з послідовністю елементів, розташованих до цієї риси.

В наступному прикладі 1 - голова списку, а [2, 3, 4, 5] - хвіст. Пролог покаже це за допомогою співставлення списку чисел зі зразком, що складається з голови й хвоста.

```
?- [1, 2, 3, 4, 5] = [Head | Tail].
Head = 1
```

```
Tail = [2, 3, 4, 5]
Yes
```

Тут Head і Tail - тільки імена змінних. Ми так само могли б використати X і Y або які-небудь інші імена змінних. Помітимо, що хвіст списку завжди є списком. Голова, у свою чергу, є елемент списку, що вірно й для всіх інших елементів, розташованих до вертикальної риси. Це дозволяє одержати, скажімо, другий елемент списку.

Приклад

Використаємо анонімні змінні для голови й списку, що стоїть після риси, якщо нам потрібний тільки другий елемент списку:

```
?- [слон, кінь, осел, собака] = [_, X | _ ].
X = кінь
Yes
```

Розглянемо кілька процедур обробки списків. Зверніть увагу, що всі вони використовують рекурсію, у якій термінальне (базове) правило визначене для порожнього списку.

Приклад

Напишемо предикат для обчислення суми всіх елементів списку чисел.

```
сума_списку([],0).
сума_списку([H|T],S):- number(H), сума_списку(T,S1),
                       S is S1+H.
```

Приклад

Предикат **місце** успішний, якщо третій аргумент є список, який отриманий вставкою першого аргументу в довільне місце списку, що є другим аргументом.

```
місце(E, L, [E|L]).
місце(E, [H|L], [H|Y]):-  місце(E, L,Y).
```

Подивимося на результати деяких запитів, що використовують цей предикат.

```
?- місце(1,[2,3],X).
X = [1, 2, 3] ;
X = [2, 1, 3] ;
X = [2, 3, 1] ;
No

?- місце(1,L,[2,1,3]).
L = [2, 3] ;
No
```

```
?- місце(X,[2,3],[2,1,3]).
X = 1 ;
No
```

Приклад

Предикат **перестановка** видає списки, отримані перестановкою елементів свого першого аргументу.

```
перестановка([],[]).
перестановка([H|L],Z):- перестановка(L,Y), місце(H,Y,Z).
```

Приклад використання:

```
?- перестановка([a,b,c],X).
X = [a, b, c] ;
X = [b, a, c] ;
X = [b, c, a] ;
X = [a, c, b] ;
X = [c, a, b] ;
X = [c, b, a] ;
No
```

І, нарешті, приведемо правило для друку всіх можливих перестановок списку:

```
всі_перестановки(L):- перестановка(L,R), write(R), nl, fail.
```

Перша підціль предиката обчислює чергову перестановку, друкує її й переходить до останньої підцілі - **fail**. Ця підціль завжди неуспішна, що змушує Пролог повернутися до початку правила й продовжити пошук рішення. Робота процедури завершиться, коли всі перестановки будуть вичерпані:

```
?- всі_перестановки(['маркіза', 'ваші прекрасні очі',
|            'обіцяють мені смерть від любові']).

[маркіза, ваші прекрасні очі, обіцяють мені смерть від любові]
[ваші прекрасні очі, маркіза, обіцяють мені смерть від любові]
[ваші прекрасні очі, обіцяють мені смерть від любові, маркіза]
[маркіза, обіцяють мені смерть від любові, ваші прекрасні очі]
[обіцяють мені смерть від любові, маркіза, ваші прекрасні очі]
[обіцяють мені смерть від любові, ваші прекрасні очі, маркіза]

No
```

Приклад

У давньояпонському календарі був прийнятий 60-річний цикл, що складається з п'яти 12-річних підциклів. Підцикли позначалися назвами

кольорів: зелений, червоний, жовтий, білий і чорний. Усередині кожного підцикла роки носили назви тварин: пацюк, корова, тигр, заєць, дракон, змія, кінь, вівця, мавпа, курка, собака й свиня. Наприклад, 1984 рік - рік початку чергового циклу - називався Роком Зеленого Пацюка.

Складемо програму, що по заданому номеру року нашої ери n друкує його назву в давньояпоському календарі. Розглянемо два випадки:

(1) значення n не менше, ніж 1984;

(2) значення n - будь-яке натуральне число.

Скористаємося вбудованим предикатом **nth0(індекс, список, елемент)**, що буде успішним, якщо елемент перебуває на місці з номером *індекс*, вважаючи від 0. Для випадку (1) використаємо предикат **nam**, для випадку (2) предикат - **nm**.

```
color(N,X):- N1 is ((N-1984) mod 60)//12,
             nth0(N1, ['зелений',
                       'червоний', 'жовтий',
                       'білий', 'чорний'],
                  X).

animal(N,X):- N1 is (N-1984) mod 12,
              nth0(N1,
                   ['пацюк',  'корова', 'тигр',
                'заєць',   'дракон', 'змія',
                     'кінь', 'вівця',   'мавпа',
                     'курка', 'собака', 'свиня'],
                  X).

nam(N,[X,Y]):- number(N), color(N,X), animal(N,Y).

nm(N,X):- N>1983, nam(N,X).
nm(N,X):- N<1984, N1 is N+60, nm(N1,X).
```

Частина 2. ЛАБОРАТОРНІ РОБОТИ

Лабораторна робота 1.

НАПИСАННЯ ПРОСТОЇ ПРОГРАМИ НА МОВІ GNU-PROLOG

Мета роботи: отримання практичних навичок складання, доопрацювання та виконання простої програми в системі програмування GNU-PROLOG.

Завдання:

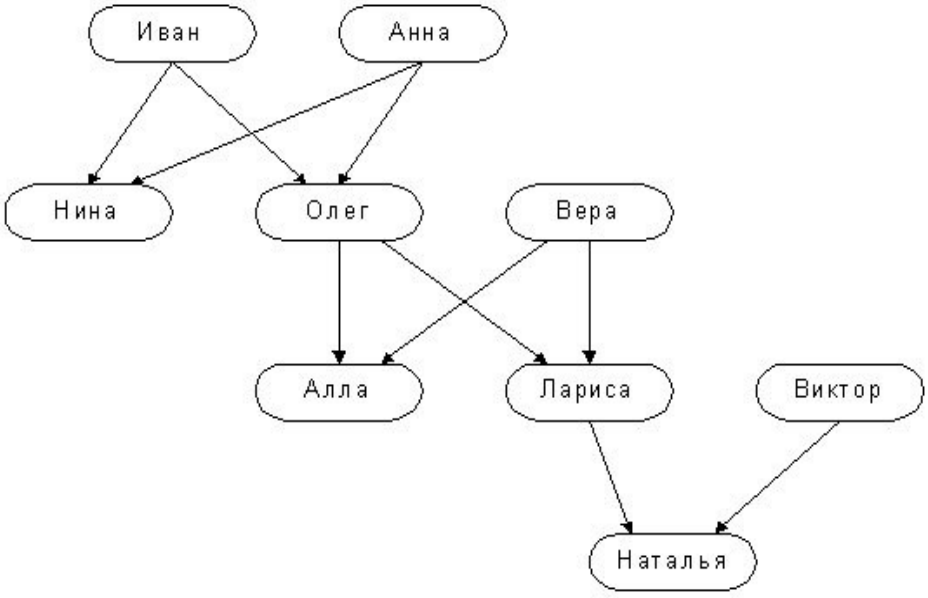
1. Проінсталювати на власному комп'ютері систему програмування GNU-PROLOG та систему редагування текстів программ SciTE (Science Text Editor).

2. Скласти на мові Prolog дерево родинних відношень, використовуючи предикат **roditel** з двома параметрами: ім'я одного з батьків та ім'я дитини. Написати на мові Prolog та запустити наступні запити:

•“Хто є і батьками, і має батьків”

•“Хто не має дітей”

Наприклад: для схеми родинних зв'язків



програма буде мати вигляд:

```
roditel(`иван`,`нина`).
roditel(`иван`,`олег`).
```

```
roditel(`анна`,`нина`).
roditel(`анна`,`олег`).
roditel(`олег`,`лариса`).
roditel(`олег`,`алла`).
roditel(`вера`,`алла`).
roditel(`вера`,`лариса`).
roditel(`лариса`,`наталья`).
roditel(`виктор`,`наталья`).
```

3. Склад звіту про виконання лабораторної роботи:

•Назва, мета та завдання лабораторної роботи

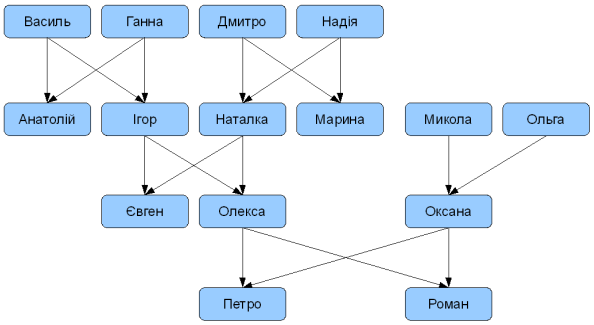
•Зміст індивідуального завдання

•Графічне зображення родинних зв'язків з предикатом

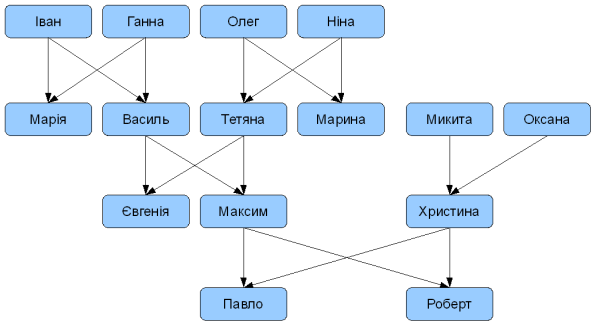
•Програма на мові Prolog

•Скріншоти виконання програми та запитів на завантажених правилах

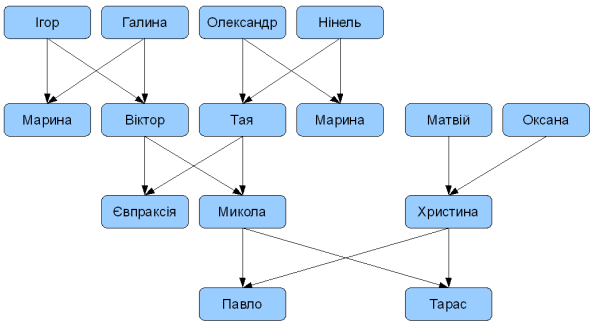
Індивідуальні завдання до лабораторної роботи 1.



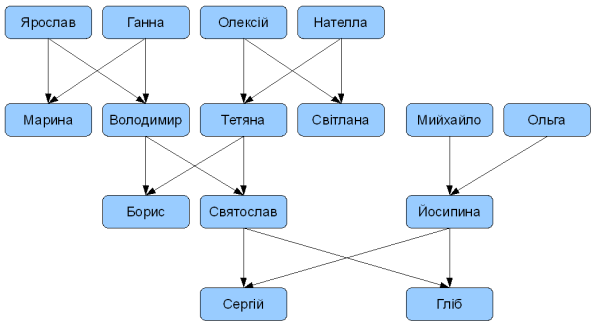
Варіант 1



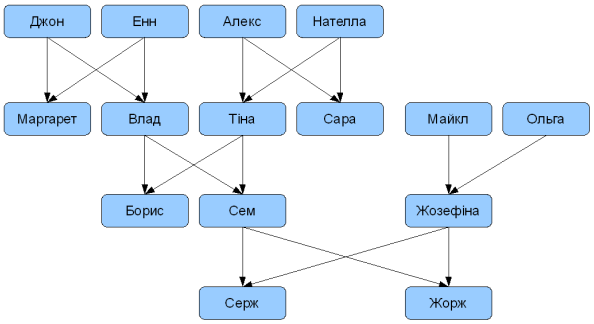
Варіант 2



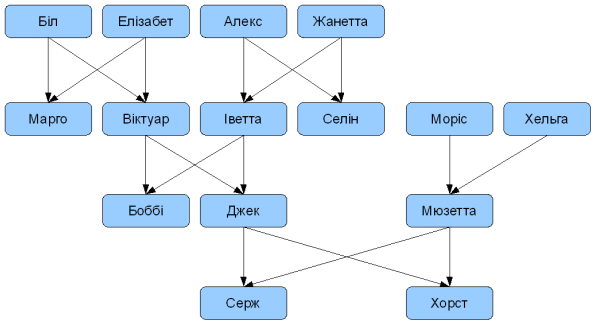
Варіант 3



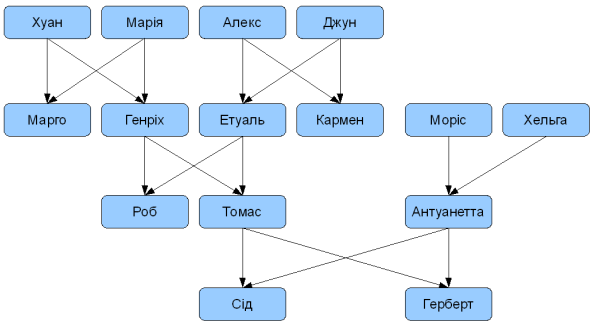
Варіант 4



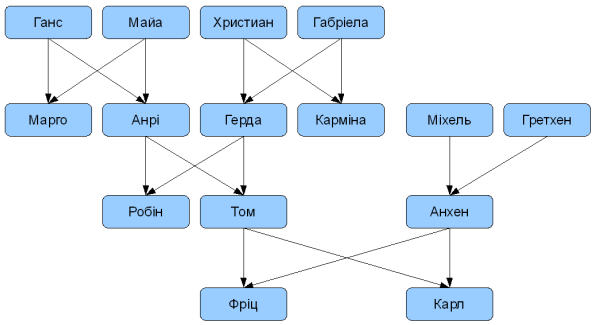
Варіант 5



Варіант 6

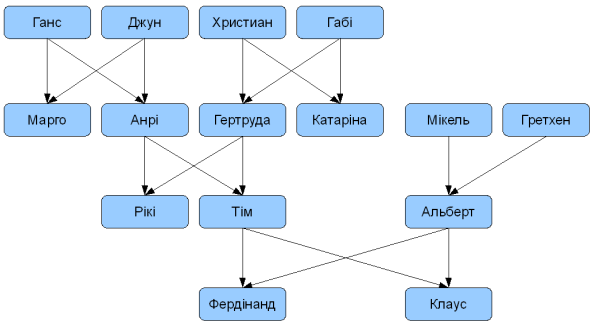


Варіант 5

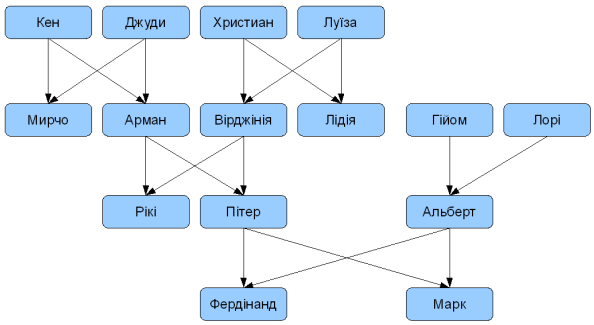


Варіант 6

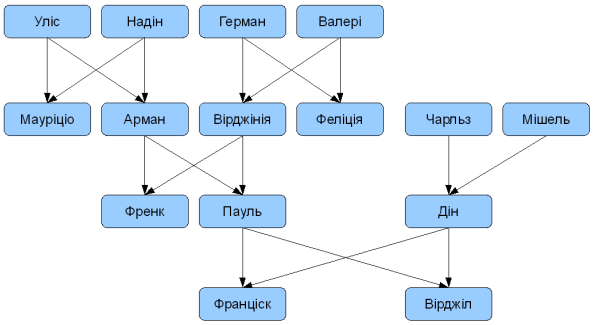
Варіант 7



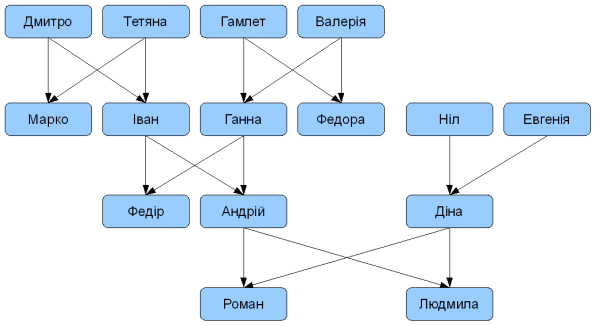
Варіант 8



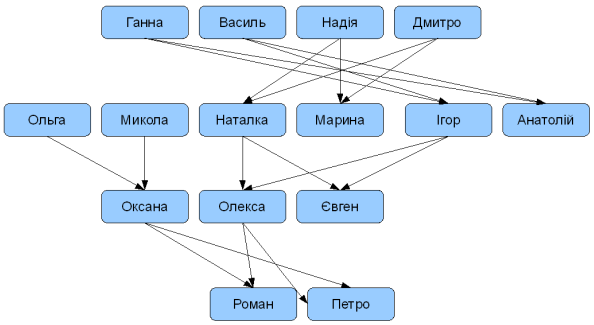
Варіант 9



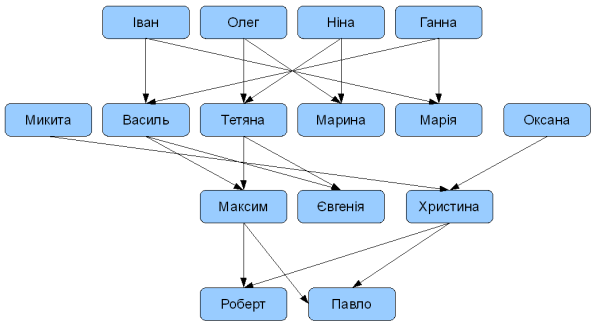
Варіант 10



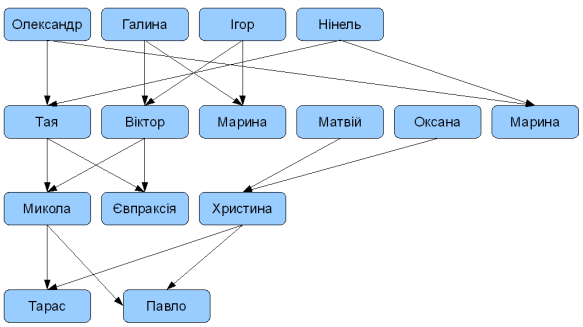
Варіант 11



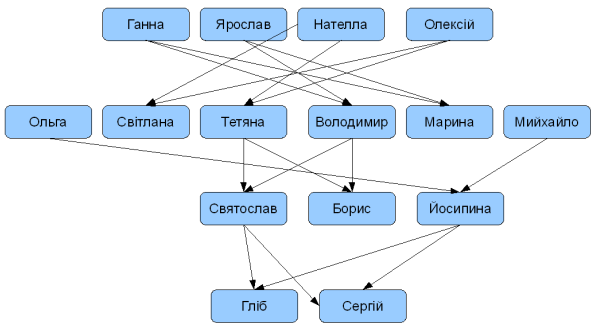
Варіант 12



Варіант 13

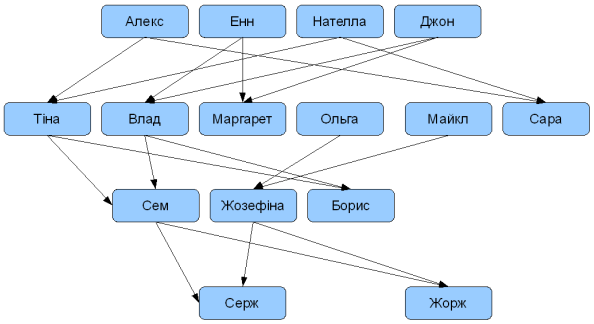


Варіант 14

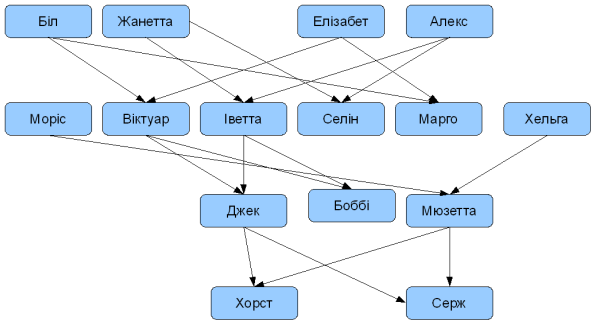


Варіант 15

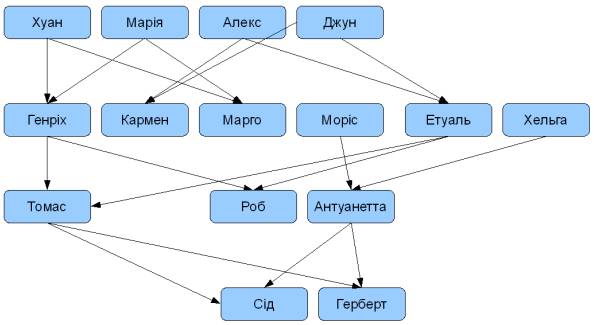
Варіант 16



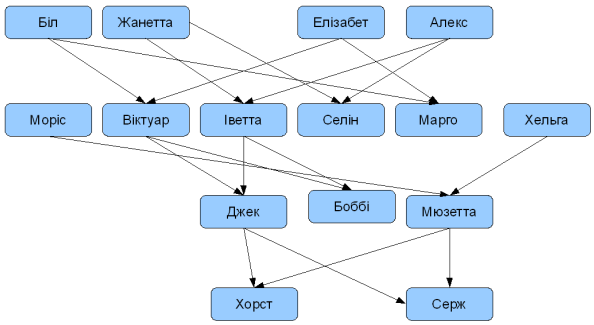
Варіант 17



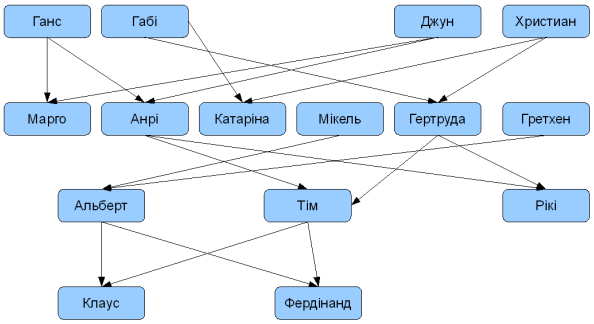
Варіант 18



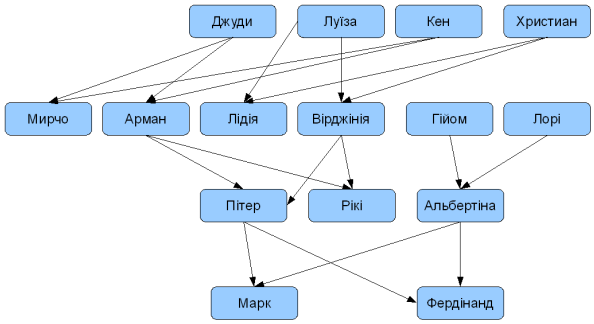
Варіант 19



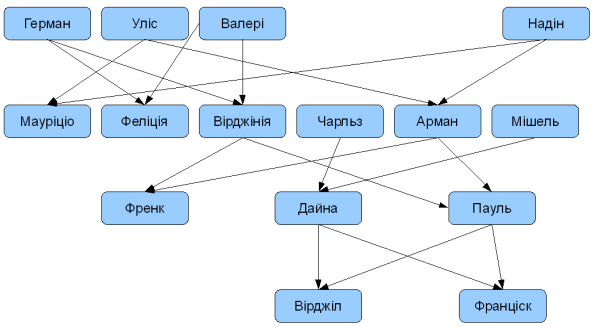
Варіант 20



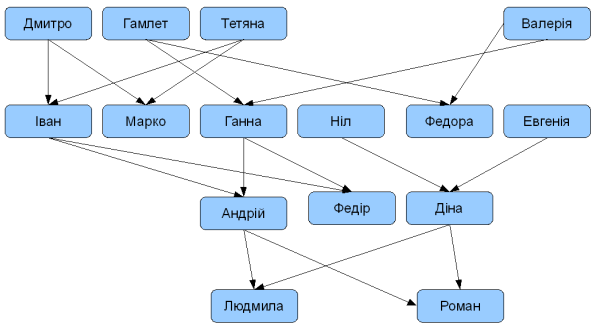
Варіант 21



Варіант 22



Варіант 23



Варіант 24

Лабораторна робота 2.

ФОРМУВАННЯ ПРАВИЛ

Мета роботи: отримання практичних навичок складення правил та використання їх в програмі в системі програмування GNU-PROLOG.

Завдання:

1. Програму з лабораторної роботи 1 доповнити новими фактами, що дозволяють побудувати правила для визначення наступних цілей-предикатів:

- батько
- мати
- син
- дочка
- брат
- сестра
- дядько
- тітка
- дід
- баба
- онук
- онучка
- небіж
- небога

2. Склад звіту про виконання лабораторної роботи:

- Назва, мета та завдання лабораторної роботи
- Зміст індивідуального завдання
- Графічне зображення родинних зв'язків з відповідними предикатами
- Програма на мові Prolog
- Скріншоти виконання програми та запитів на завантажених правилах

Лабораторна робота 3.

РЕКУРСІЯ

Мета роботи: отримання практичних навичок складення та доопрацювання програм з використанням рекурсії.

Завдання:

1. Підсумувати цілі позитивні числа, які йдуть одне за другим з кроком **d**, та закінчуються числом **n**. Значення **d** та **n** вводяться за запитом з екрану монітора (наприклад, **d=3** та **n=11**, підсумок **11+8+5+2=26**). У випадку **d>=n** підсумок дорівнює **n**.

2. Звести число **a** у цілий ступінь **m** (**m** може бути позитивним, нульовим або негативним). Скласти два варіанти програми:

а) використовуючи рекурсивний вираз **a^m=a^(m-1)*a**;

б) використовуючи можливість обчислень за формулою **a^(2*m)=a^m*a^m** для парного ступеня.

Визначити максимальний ступінь, у який можна звести число **a=2** по першому й другому варіантах програми. Результати й причини їхнього розходження відобразити у висновках звіту.

3. Знайти підсумок цілих послідовних чисел від 0 до **N**.

4. Обчислити значення наступних функцій, використовуючи розкладання в ряд (в ітеративному варіанті програми задавати точність обчислення функції):

$$\pi^2/6 = 1 + 1/2^2 + 1/3^2 + ... + 1/k^2 + ...$$

5. Склад звіту про виконання лабораторної роботи:

•Назва, мета та завдання лабораторної роботи

•Зміст індивідуального завдання

•Програми на мові Prolog

•Скріншоти виконання програм та запитів на завантажених правилах

Індивідуальні завдання до лабораторної роботи 3.

Варіант	d	n	a	m	N	Варіант	d	n	a	m	M
1	3	100	0,6795	3	20	13	3	105	0,0036	3	21
2	4	110	0,9814	2	22	14	4	115	0,5232	2	23
3	5	120	0,9951	0	24	15	5	125	0,2187	0	25
4	6	130	0,3367	-2	26	16	6	135	0,9969	-2	27
5	3	140	0,3323	-3	28	17	3	145	0,8010	-3	29
6	4	150	0,2003	-4	30	18	4	155	0,3965	-4	31
7	5	160	0,6957	4	32	19	5	165	0,4248	4	33
8	6	170	0,9857	5	34	20	6	175	0,6734	5	35
9	3	180	0,1553	6	36	21	3	185	0,6073	6	37
10	4	190	0,4601	0	38	22	4	195	0,0694	0	39
11	5	200	0,8551	-5	40	23	5	205	0,8069	-5	41
12	6	210	0,0747	-6	42	24	6	215	0,2511	-6	43

Лабораторна робота 4.

ВИКОРИСТАННЯ ВІДСІКАННЯ У ПРОЛОГ-ПРОГРАМАХ

Мета роботи: отримання практичних навичок використання відсікання у програмах.

Завдання:

1. Визначити віковий статус людини за відомим роком народження у відповідності до таблиці. Розробити два варіанти: без відсікання та з його використанням.

2. Склад звіту про виконання лабораторної роботи:

•Назва, мета та завдання лабораторної роботи

•Зміст індивідуального завдання

•Програми на мові Prolog

•Скріншоти виконання програм та запитів на завантажених правилах

Індивідуальні завдання до лабораторної роботи 4.

Варіант	Немовля		Дитина		Підліток		Юнак		Чоловік		Старий		Довгожитель	
	від	до	від	до	від	до	від	до	від	до	від	до	від	до
1	0	1	1	11	11	15	15	21	21	65	65	90	90	
2	0	2	2	12	12	16	16	22	22	66	66	91	91	
3	0	3	3	13	13	17	17	23	23	67	67	92	92	
4	0	1	1	14	14	18	18	24	24	68	68	93	93	
5	0	2	2	11	11	19	19	25	25	69	69	94	94	
6	0	3	3	12	12	15	15	26	26	70	70	95	95	
7	0	1	1	13	13	16	16	21	21	71	71	96	96	
8	0	2	2	14	14	17	17	22	22	72	72	97	97	
9	0	3	3	11	11	18	18	23	23	73	73	98	98	
10	0	1	1	12	12	19	19	24	24	74	74	99	99	
11	0	2	2	13	13	15	15	25	25	75	75	100	100	
12	0	3	3	14	14	16	16	26	26	65	65	90	90	
13	0	1	1	11	11	17	17	21	21	66	66	91	91	
14	0	2	2	12	12	18	18	22	22	67	67	92	92	
15	0	3	3	13	13	19	19	23	23	68	68	93	93	
16	0	1	1	14	14	15	15	24	24	69	69	94	94	
17	0	2	2	11	11	16	16	25	25	70	70	95	95	
18	0	3	3	12	12	17	17	26	26	71	71	96	96	
19	0	1	1	13	13	18	18	21	21	72	72	97	97	
20	0	2	2	14	14	19	19	22	22	73	73	98	98	
21	0	3	3	11	11	15	15	23	23	74	74	99	99	
22	0	1	1	12	12	16	16	24	24	75	75	100	100	
23	0	2	2	13	13	17	17	25	25	65	65	90	90	
24	0	3	3	14	14	18	18	26	26	66	66	91	91	

Лабораторна робота 5.

РОБОТА ЗІ СПИСКАМИ

Мета роботи: отримання практичних навичок роботи зі списками у програмах.

Завдання:

1. Зі списку **L1** отримати список **L2**, черговий елемент якого дорівнює середньому арифметичному чергової трійки елементів списку **L1** . Якщо кількість елементів **L1** не кратна **3**, то останній елемент списку **L2** отримується діленням на 3 одного або підсумку двох останніх елементів списку **L1** . Список **L1** вводиться за підказкою з екрану. У підсумку виконання програми повинні виводитися вхідний **L1** та результуючий **L2** списки.

2. Провести циклічний зсув елементів списку на **n** позицій у напрямку(**LR**). Кількість елементів у списку — **m**. Елементи списку вводяться за запитом.

3. Склад звіту про виконання лабораторної роботи:

•Назва, мета та завдання лабораторної роботи

•Зміст індивідуального завдання

•Програми на мові Prolog

•Скріншоти виконання програм та запитів на завантажених правилах

Індивідуальні завдання до лабораторної роботи 5.

Варіант	n	m	LR	Варіант	n	m	LR
1	1	30	вліво	13	1	30	вправо
2	2	31	вліво	14	2	31	вправо
3	3	32	вліво	15	3	32	вправо
4	4	33	вліво	16	4	33	вправо
5	5	34	вліво	17	5	34	вправо
6	6	35	вліво	18	6	35	вправо
7	7	36	вліво	19	7	36	вправо
8	8	37	вліво	20	8	37	вправо
9	9	38	вліво	21	9	38	вправо
10	10	39	вліво	22	10	39	вправо
11	11	40	вліво	23	11	40	вправо
12	12	41	вліво	24	12	41	вправо