

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна

В. Є. Стрілець
С. І. Шматков

СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

Практикум

Електронне видання

Харків – 2022

УДК 004.89+004.4'2
С 83

Рецензенти:

Г. М. Жолткевич – доктор технічних наук, професор, декан факультету математики і інформатики Харківського національного університету імені В. Н. Каразіна;

І. В. Гребеннік – доктор технічних наук, професор, завідувач кафедри системної техніки Харківського національного університету радіоелектроніки.

*Затверджено до розміщення в мережі Інтернет рішенням Науково-методичної ради
Харківського національного університету імені В. Н. Каразіна
(протокол № 6 від 16 лютого 2022 року)*

Стрілець В. Є.

С 83 Системи штучного інтелекту : практикум [Електронне видання] / В. Є. Стрілець, С. І. Шматков. – Харків : ХНУ імені В. Н. Каразіна, 2022. – (PDF 68 с.)

У практикумі викладаються основи програмування на мові Пролог, яка використовується для розв'язання задач штучного інтелекту й обробки складних символьних структур. Описані методи і засоби логічного програмування мовою SWI-Prolog, наведені приклади пролог-програм. Посібник містить методичні вказівки до виконання практичних робіт з дисципліни «Системи штучного інтелекту», яка викладається у рамках підготовки бакалаврів за спеціальностями 123 «Комп'ютерна інженерія» та 151 «Автоматизація та комп'ютерно-інтегровані технології».

УДК 004.89+004.4'2

© Харківський національний університет імені В. Н. Каразіна, 2022

© Стрілець В. Є., Шматков С. І., 2022

© Дончик І. М., макет обкладинки, 2022

Електронне навчальне видання
комбінованого використання
Можна використовувати в локальному та мережному режимі

Стрілець Вікторія Євгенівна
Шматков Сергій Ігорович

СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

Практикум

Електронне видання

Коректор *Л. Є. Стешенко*
Комп'ютерне верстання *В. В. Савінкова*
Макет обкладинки *І. М. Дончик*

Підписано до видання 16.05.2023. Гарнітура Times New Roman
Обсяг 2,26 Мб. Зам. 31/22.

Харківський національний університет імені В. Н. Каразіна,
61022, м. Харків, майдан Свободи, 4.
Свідоцтво суб'єкта видавничої справи ДК № 3367 від 13.01.2009

Видавництво ХНУ імені В. Н. Каразіна

Зміст



Вступ	4
Практична робота № 1. Числення висловлювань	6
Практична робота № 2. Предикати, факти і правила в SWI-Prolog	10
Практична робота № 3. Рекурсія в SWI-Prolog	20
Практична робота № 4. Списки у SWI-Prolog	26
Практична робота № 5. Робота з базами даних у SWI-Prolog	30
Практична робота № 6. Пошук у просторі станів	36
Практична робота № 7. Віконні додатки у SWI-Prolog	47
Додатки	53
Додаток А. Початок роботи у SWI-Prolog. Найпростіша програма	53
Додаток Б. Приклади Prolog-програм	58
Додаток В. Вбудовані предикати і оператори SWI-Prolog	64
Список літератури	67

Вступ



Штучний інтелект (англ. Artificial intelligence) – наука та технологія створення інтелектуальних машин, в особливості інтелектуальних комп'ютерних програм. Штучний інтелект пов'язаний з завданням використання комп'ютерів для розуміння людського інтелекту, але не обов'язково обмежується біологічно правдоподібними методами (Джон Маккарті, 1956 р., конференція у Дартмутському університеті). В подальшому було зроблено чимало спроб дати формальне визначення інтелекту взагалі та інтелекту штучного зокрема.

Найбільш відомим є визначення предмета теорії штучного інтелекту, що було дане видатним дослідником у галузі штучного інтелекту М. Мінські і яке у більш або менш видозміненому вигляді потрапило до словників та енциклопедій: «штучний інтелект є дисципліна, що вивчає можливість створення програм для вирішення задач, які при вирішенні їх людиною потребують певних інтелектуальних зусиль». Це визначення зустріло критику, яка полягала в тому, що під нього можна підвести що завгодно, наприклад, виконання простих арифметичних операцій. Відтак до цього визначення додається поправка: «сюди не входять задачі, для яких відома процедура їх вирішення».

Єдиної відповіді на питання, чим займається штучний інтелект, не існує. Майже кожен автор дає своє визначення. Зазвичай ці визначення зводяться до таких:

- штучний інтелект вивчає методи розв'язання задач, які потребують людського розуміння. Тут мова іде про те, щоб навчити штучний інтелект розв'язувати тести інтелекту. Це передбачає розвиток способів розв'язання задач за аналогією, методів дедукції та індукції, накопичення базових знань і вміння їх використовувати;

- штучний інтелект вивчає методи розв'язання задач, для яких не існує способів розв'язання або вони не коректні (через обмеження в часі, пам'яті тощо). Завдяки такому визначенню інтелектуальні алгоритми часто використовуються для розв'язання NP-повних задач, наприклад, задачі комівояжера;

- штучний інтелект займається моделюванням людської вищої нервової діяльності;

- штучний інтелект – це системи, які можуть оперувати знаннями, а найголовніше – навчатися. В першу чергу мова йде про те, щоби визнати клас експертних систем (назва походить від того, що вони спроможні замінити «на посту» людей-експертів) інтелектуальними системами.

Останнє визначення з'явилося у 1990-х рр. і засноване на так званому агентно-орієнтованому підході. Цей підхід акцентує увагу на тих методах і алгоритмах, які допоможуть інтелектуальному агенту виживати в оточуючому середовищі під час виконання свого завдання. Тому тут значно краще застосовуються алгоритми пошуку і прийняття рішення.

Для створення систем штучного інтелекту були розроблені мови програмування, найбільш відомими є Lisp і Prolog.

Prolog – мова логічного програмування загального призначення, яка має корені в логіці першого порядку, математичній логіці, та, на відміну від багатьох інших мов програмування, є декларативною: логіка програми виражається в термінах відношень, представлених як факти та правила. Обчислення ініціюються запуском запиту над цими відношеннями.

Першу систему Prolog було розроблено в 1972 р. Аланом Кольмером та Філіпом Русселем.

Prolog була однією з перших логічних мов програмування, й залишається найпопулярнішою серед таких мов і на сьогодні, маючи декілька безкоштовних та комерційних реалізацій. Її застосовували як для доведення теорем, експертних систем, так і для її початкової області призначення, обробки природної мови. Сучасні середовища Prolog підтримують як створення графічних інтерфейсів користувача, так і адміністративні або мережеві застосування.

Prolog добре підходить для розв'язання специфічних задач, що виграють від логічних запитів на базі правил, таких як пошук базами даних, системи голосового керування та заповнення шаблонів.

У межах дисципліни «Системи штучного інтелекту» студенти мають отримати навички розв'язання задач штучного інтелекту з використанням мови Prolog. Теми і структура практичних робіт розроблені таким чином, щоб допомогти студентам оволодіти навичками програмування на Prolog, починаючи з основ.

За результатами вивчення дисципліни студенти повинні вміти конструювати алгоритми розв'язання задач на основі числення предикатів; використовувати системи, що ґрунтуються на правилах, для побудови і модифікації баз знань; розв'язувати задачі неінформованого та евристичного пошуку, в тому числі з використанням мови логічного програмування Prolog.

Практична робота № 1

Числення висловлювань



Мета роботи: пригадати елементи теорії логіки висловлювань; навчитися подавати висловлювання у вигляді формул алгебри висловлювань та будувати для них таблиці істинності.

Теоретичні відомості

Вихідним поняттям теорії логіки є висловлювання. Будь-яке розповідне речення, яке може бути істинним або брехливим, називають *висловлюванням*. Логічним значенням висловлювань є «істина» (true) або «брехня» (false).

Приклади розповідних речень, які є висловлюваннями: «Трійка є простим числом» (має значення «істина»), «3.14 – раціональне число» (має значення «брехня»), «Х. Колумб відкрив Америку» (істинне), «Київ – столиця Молдови» (брехня).

Такі висловлювання є *простими* або елементарними. При формальному розв'язанні задач замість поняття «прості висловлювання» використовують поняття «пропозиційні змінні» (від лат. *propositio* – пропозиція), які позначають великими літерами латинського алфавіту А, В, С, D і т. д. Істинність або брехливість висловлювань позначають символами Т (істина, true) і F (брехня, false).

Наприклад:

- якщо $A = \text{«Трійка є простим числом»}$, то $A = T$;
- якщо $B = \text{«3.14 – раціональне число»}$, то $B = F$;
- якщо $C = \text{«Пінгвіни живуть в Арктиці»}$, то $C = F$;
- якщо $D = \text{«Відень – столиця Австрії»}$, то $D = T$.

Висловлювання, які отримують із речень за допомогою граматичних зв'язок «не», «і», «або», «якщо ..., то ...», «... тоді і лише тоді, коли ...» та ін., називають *складними* або складеними. Для позначення граматичних зв'язок використовують логічні (або пропозиційні) зв'язки: \wedge – «і», \vee – «або», \neg – «не», \equiv – еквівалентність, \rightarrow – «якщо ..., то ...» (наслідок).

Наприклад:

- складне висловлювання «Трійка є простим і цілим числом» можна подати як об'єднання простих висловлювань $A_1 = \text{«Трійка є простим числом»}$ і $A_2 = \text{«Трійка є цілим числом»}$, тоді $A_1 \wedge A_2 = T$;
- складне висловлювання «Якщо число 11 є простим, то воно ціле» можна подати як наслідок простих висловлювань $B_1 = \text{«Число 11 є простим»}$ і $B_2 = \text{«Число 11 є цілим»}$, тобто $B_1 \rightarrow B_2 = T$.

Правила побудови складних висловлювань як послідовності пропозиційних змінних, логічних зв'язок і допоміжних символів визначають можливість формального розв'язання різних задач.

Правила виконання логічних операцій над складними висловлюваннями на основі визначених логічних зв'язок і пропозиційних змінних формують *алгебру висловлювань*.

Правила виводу нових висловлювань, заснованих на відомих відношеннях між заданими пропозиційними змінними, формують *числення висловлювань*. Висловлювання, з яких роблять вивід нових висловлювань, називають посиленням, а отримуване висловлювання – висновком.

Символами числення висловлювань, тобто алфавітом, є логічні константи true і false, які позначають Т і F, пропозиційні змінні А, В, С, ..., які позначають літерами латинського алфавіту, логічні зв'язки \wedge – кон'юнкція, \vee – диз'юнкція, \neg – заперечення, \equiv – еквівалентність, \rightarrow – імплікація і круглі дужки.

Будь-яке складне висловлювання, яке можна отримати з елементарних висловлювань із застосуванням логічних зв'язок, називають *формулою* числення висловлювань.

Будь-яку пропозиційну змінну можна вважати формулою нульового порядку.

Якщо F_1 і F_2 – формули, то $\neg F_1$, $\neg F_2$, $F_1 \wedge F_2$, $F_1 \vee F_2$, $F_1 \equiv F_2$, $F_1 \rightarrow F_2$ також є формулами. Ніяких інших формул у численні висловлювань немає.

Значення формули повністю визначається значеннями пропозиційних змінних, які до неї входять.

Всі значення формули у залежності від значень елементарних висловлювань, що до неї входять, можуть бути повністю описану за допомогою *таблиці істинності*. У табл. 1 наведені значення логічних зв'язків, які використовуються у численні висловлювань.

Таблиця 1

Значення основних логічних операцій

X	Y	$\neg X$	$X \wedge Y$	$X \vee Y$	$X \rightarrow Y$	$X \equiv Y$
F	F	T	F	F	T	T
F	T	T	F	T	T	F
T	F	F	F	T	F	F
T	T	F	T	T	T	T

Приклади застосування логічних операцій:

– є висловлювання A =«Комп'ютер містить мікропроцесор», B =«Комп'ютер містить оперативну пам'ять», C =«Комп'ютер містить контролери» і D =«Комп'ютер містить порти вводу-виводу». Тоді складене висловлювання «Комп'ютер містить мікропроцесор, оперативну пам'ять, контролери і порти вводу-виводу» можна записати формулою $F=A \wedge B \wedge C \wedge D$;

– є висловлювання A =«По провіднику тече електричний струм» і B =«Навколо провідника є магнітне поле». Тоді формула $F=A \rightarrow B$ відображує висловлювання «Якщо по провіднику протікає електричний струм, то навколо провідника виникає магнітне поле»;

– є висловлювання A = «Бути парним числом» і B = «Число ділиться на два», тоді формула $F=(A \equiv B)$ відображає висловлювання «Для того, щоб число було парним, необхідно і достатньо, щоб воно ділилося на два».

Для визначення істинності складного висловлювання необхідно проаналізувати значення істинності кожного висловлювання, яке входить до його складу, і сформулювати послідовно значення істинності кожної підформули, яка входить у формулу складного висловлювання. Логічні значення формули алгебри логіки повністю визначається логічними значеннями пропозиційних змінних, які входять до неї. Усі можливі логічні значення формули у залежності від значень елементарних висловлювань, які до неї входять, можуть бути повністю описані у таблиці істинності.

Розглянемо як приклад складне висловлювання «Якщо обчислювальна задача немає точного розв'язку, то використовують чисельні методи або розробляють нові підходи, а якщо використовують чисельні методи, то отримують наближений розв'язок, і, нарешті, якщо є наближений розв'язок, при цьому задача немає точного, то розробка нових підходів не потрібна».

У цьому висловлюванні є чотири елементарні висловлювання, які можна замінити пропозиційними змінними: A = «Обчислювальна задача немає точного розв'язку», B = «Використовують чисельні методи», C = «Розробляють нові підходи» і D = «Отриманий наближений розв'язок»; і записати формулу висловлювання:

$$F = (A \rightarrow (B \vee C)) \& (B \rightarrow D) \& ((D \wedge A) \rightarrow \neg C).$$

Істинність формули F для різних значень істинності пропозиційних змінних A, B, C, D можна визначити склавши для неї таблицю істинності (табл. 2).

Таблиця 2

Таблиця істинності для складного висловлювання

A	B	C	D	$\neg C$	$D \& A$	$B \vee C$	$1 \rightarrow 7$	$B \rightarrow D$	$6 \rightarrow 5$	$8 \& 9$	$11 \& 10$
1	2	3	4	5	6	7	8	9	10	11	12
F	F	F	F	T	F	F	T	T	T	T	T
F	F	F	T	T	F	F	T	T	T	T	T
F	F	T	F	F	F	T	T	T	T	T	T
F	F	T	T	F	F	T	T	T	T	T	T
F	T	F	F	T	F	T	T	F	T	F	F
F	T	F	T	T	F	T	T	T	T	T	T
F	T	T	F	F	F	T	T	F	T	F	F
F	T	T	T	F	F	T	T	T	T	T	T
T	F	F	F	T	F	F	F	T	T	F	F
T	F	F	T	T	T	F	F	T	T	F	F
T	F	T	F	F	F	T	T	T	T	T	T
T	F	T	T	F	T	T	T	T	F	T	F
T	T	F	F	T	F	T	T	F	T	F	F
T	T	F	T	T	T	T	T	T	T	T	T
T	T	T	F	F	F	T	T	F	T	F	F
T	T	T	T	F	T	T	T	T	F	T	F

Завдання для практичної роботи

1. Для логічного виразу (формули) побудувати таблицю істинності (за варіантами).

2. Записати складнопідрядне речення, виділити в ньому елементарні висловлювання (пропозиційні змінні), записати формулу, яка відповідає реченню, та для формули побудувати таблицю істинності.

Кількість елементарних висловлювань у реченні має бути не менше трьох.

Варіанти для завдання 1

1. $(a \vee (\neg d \wedge b)) \wedge ((\neg a \wedge (\neg b \vee d)) \vee c) \vee \neg c \vee (a \vee (b \wedge \neg d))$.
2. $((a \vee c) \wedge (a \vee d)) \wedge (((c \vee (c \wedge b)) \wedge \neg c) \vee \neg a)$.
3. $(\neg b \vee d) \wedge ((\neg d \wedge c) \vee (a \wedge c) \vee (\neg d \wedge \neg c)) \wedge (b \vee d)$.
4. $(a \vee \neg c) \wedge (\neg a \vee \neg b) \wedge (\neg b \vee c) \wedge (\neg a \vee b) \wedge (b \vee c)$.
5. $(a \wedge c) \vee ((b \vee \neg d) \wedge (\neg a \vee \neg d) \wedge (d \vee b) \wedge (\neg a \vee d)) \vee (a \wedge \neg c)$.
6. $((\neg b \vee \neg c) \wedge (a \vee b)) \vee (d \wedge \neg c) \vee (((\neg b \wedge \neg a) \vee c) \wedge (a \vee b))$.
7. $(a \wedge \neg c) \vee (\neg a \wedge \neg b) \vee (b \wedge c) \vee (\neg a \wedge b) \vee (c \wedge \neg b)$.
8. $(a \vee (c \vee (b \wedge c))) \wedge \neg(c \wedge d) \wedge (c \wedge \neg d) \wedge (c \vee (\neg d \wedge \neg c) \vee d)$.
9. $((a \vee \neg c) \wedge (\neg b \vee \neg d) \wedge (\neg b \vee \neg c) \wedge (\neg c \vee d)) \vee ((\neg b \vee c) \wedge (c \vee d))$.
10. $(a \vee \neg c) \wedge ((\neg a \wedge d) \vee (b \wedge d) \vee (\neg a \wedge \neg d) \vee (b \wedge \neg d)) \wedge (a \vee c)$.
11. $((d \wedge \neg c) \vee (\neg d \wedge \neg b) \vee (c \wedge \neg b)) \wedge ((\neg d \wedge b) \vee (c \wedge b)) \wedge (\neg a \vee b)$.
12. $((\neg c \wedge \neg d) \vee (b \wedge c)) \wedge (\neg a \vee \neg d) \wedge (((\neg c \vee \neg b) \wedge d) \vee (c \wedge b))$.
13. $((a \vee b) \wedge (\neg b \wedge c \wedge d)) \vee (\neg a \wedge \neg b \wedge c \wedge d) \vee \neg b \vee \neg c \vee d$.
14. $((a \wedge b) \vee (a \wedge \neg b)) \vee ((\neg a \vee b) \wedge (c \vee \neg d) \wedge (\neg a \vee \neg b) \wedge (d \vee c))$.
15. $((\neg b \wedge c) \vee (\neg c \vee d) \vee \neg a) \wedge (\neg a \vee b \vee \neg c \vee d) \wedge \neg(c \vee d) \wedge a$.
16. $((b \vee c) \wedge (d \vee (\neg b \wedge \neg c))) \vee (\neg d \wedge \neg a) \vee ((c \vee b) \wedge (\neg d \vee \neg c))$.
17. $((\neg c \vee d) \wedge (d \vee a)) \vee ((b \vee \neg d) \wedge (\neg c \vee \neg a) \wedge (\neg c \vee \neg d) \wedge (\neg d \vee a))$.
18. $(b \wedge d) \vee ((c \vee \neg d) \wedge (a \vee c) \wedge (\neg d \vee c) \wedge (a \vee \neg c)) \vee (\neg b \wedge d)$.
19. $(a \wedge \neg d) \vee (((\neg c \wedge \neg b) \vee d) \wedge (c \vee b)) \vee (\neg d \vee \neg c) \wedge (c \vee b)$.
20. $((d \vee (d \wedge c)) \wedge \neg d) \vee \neg b \wedge ((b \vee d) \wedge (b \vee a))$.

Контрольні питання

1. Визначення висловлювання. Елементарні висловлювання.
2. Визначення алгебри висловлювань.
3. Визначення числення висловлювань.
4. Алфавіт числення висловлювань.
5. Визначення формули числення висловлювання.
6. Правила побудови формул.
7. Таблиці істинності.

Практична робота № 2

Предикати, факти і правила в SWI-Prolog



Мета роботи: отримати навички у використанні середовища програмування SWI-Prolog для розв'язання задач в області штучного інтелекту та математики.

Теоретичні відомості

Програма мовою SWI-Prolog є набором фактів і (за потреби) правил. Якщо програма містить лише факти, то її називають *базою даних*. Якщо вона містить ще й правила, то часто використовують термін *база знань*. Програма зазвичай описує деяку дійсність. Об'єкти (елементи) дійсності, що описується, подаються за допомогою термів. *Терм* – це об'єкт. Існує чотири види термів: атоми, числа, змінні і складені терми. Атоми і числа називають простішими термами.

Атом – це окремий об'єкт, який вважається елементарним. У SWI-Prolog атом подається послідовністю літер нижнього і верхнього регістру, цифр і символу підкреслення «_», починаючи зі строкової букви.

Змінними є строки символів, цифр і символу підкреслення, які починаються з великої літери або символу підкреслення. Символ підкреслення «_» є особливим випадком – анонімною змінною (змінною без імені). Анонімна змінна використовується тоді, коли значення змінної не використовується у програмі.

Арифметичні вирази. Є у SWI-Prolog набір вбудованих атомів (функцій) для обчислення арифметичних виразів, деякі з них подані у таблиці 3.

Таблиця 3

Арифметичні функції у SWI-Prolog

$X + Y$	сума X і Y
$X - Y$	різниця між X і Y
$X * Y$	добуток X і Y
X / Y	ділення X на Y
$X \bmod Y$	остача від ділення X на Y
$X // Y$	ділення націло X на Y
$X ** Y$	піднесення X у степінь Y
$-X$	зміна знака X
$\text{abs}(X)$	абсолютна величина числа X
$\text{max}(X, Y)$	більше з чисел X і Y
$\text{min}(X, Y)$	менше з чисел X і Y
$\text{sqrt}(X)$	квадратний корінь з X
$\text{random}(\text{Int})$	випадкове ціле число в діапазоні від 0 до Int
$\sin(X)$	синус X
$\cos(X)$	косинус X
$\tan(X)$	тангенс X

Продовження табл. 3

$\log(X)$	натуральний логарифм (\ln) числа X
$\log_{10}(X)$	десятковий логарифм (\lg) числа X
$\text{float}(X)$	дійсне число, яке відповідає цілому числу X
π	3.14159 (наближене значення числа π)
e	2.71828 (наближене значення числа e)

Складені терми (функції) складаються з імені функції (нечислового атому) і списку аргументів (атомів, чисел, змінних або інших складених термів), який береться у круглі дужки і розділяється комами. Групи складених термів використовують для складання фраз. Не можна ставити символ пробілу між іменем функції (функтором) і дужками. У інших позиціях, однак, пробіли можуть бути корисними для написання більш читабельних програм. Список символів може бути поданий у вигляді строк, які беруться у лапки.

Факт – це твердження про те, що дотримується деяке конкретне співвідношення. Він безумовно є істинним. Наприклад: «Сьогодні сонячно», «Т. Г. Шевченко – український письменник». Кома між фактами означає операцію логічного «і» (кон'юнкцію), факт можна записати у вигляді предиката, аргументи якого є символічними або числовими константами.

У загальному випадку *предикат* – це логічна функція від одного або деяких аргументів, тобто функція, яка діє у множині з двох логічних значень: істина і брехня. Предикат записується у вигляді складеного терму:

ім'я_предиката (аргументи).

Аргументи перелічуються через кому і є деякими об'єктами або властивостями об'єктів, а ім'я предиката означає зв'язок або співвідношення між аргументами. Предикат однозначно визначається парою: ім'я і кількість аргументів. Два предикати з однаковим ім'ям, але різною кількістю аргументів, вважаються різними.

Кількість параметрів предиката називається його *арністю*. Під час опису предиката арність указують після його імені, розділяючи їх символом слешу «/».

Порядок аргументів предиката пов'язаний із сенсом факту, і тому його змінювати неможна.

Під час запису фактів (предикатів) необхідно пам'ятати: ім'я факту починається зі строкової літери; запис кожного факту закінчується крапкою.

База даних – це сукупність фактів. У процесі роботи у базу даних можна додавати нові факти, видаляти або змінювати існуючі.

Наприклад, опишемо факт того, що дехто є батьками когось:

```
parent(Mary, Alex). % Mary є одним з батьків Alex
parent(George, Alex).
woman(Mary). % Mary – жінка.
man(George). % George – чоловік.
man(Alex).
```

Тепер можна сформулювати питання:

?- parents(Mary, Alex).

Yes

?- parents(George, Vita).

No.

?- woman(Alex).

No.

Запит – це послідовність предикатів, які розділені комами і завершуються крапкою. За допомогою запитів можна «запитувати» у бази даних про те, які твердження є істинними. Предикат запиту називається ціллю. Використання змінних у запитах дозволяє ставити більш складні питання.

Наприклад, питання «Хто є батьками Alex?»:

?- parent(X, Alex).

X = Mary.

X = George.

Крім фактів, програми мовою SWI-Prolog можуть мати *правила*, які дозволяють отримувати додаткові знання про той світ, який описує програма. Правило задає новий предикат через визначені раніше. Правило складається з голови (предиката) і тіла (послідовності предикатів, які розділені комами). Голова і тіло розділяються знаком «:-» і, як і для кожного факту, правило повинно закінчуватись крапкою. Знак «:-» є схематичним записом стрілки «←» і показує, що з правої частини виходить ліва. Цей знак можна прочитати як «якщо». Інтуїтивний сенс правила полягає у тому, що мета, яка є головою, буде істинною, якщо можна показати, що всі вирази (підцілі) у тілі правила є істинними.

Приклад 1. Скласти предикат, який буде визначати максимальне з двох чисел.

Розв'язок

У предикаті буде три аргументи: перші два – вхідні числа, третій – вихідний, у який буде записуватись максимум серед двох перших аргументів.

Запишемо, що у випадку, якщо перше число більше другого, максимальним є перше число, у випадку, коли перше число менше, максимумом буде друге число. Також треба не забути про ситуацію, коли числа дорівнюють одне одному, в цьому випадку максимумом буде будь-яке з них.

Розв'язок можна записати так:

max(X, Y, X) :- X > Y. /*якщо перше число більше другого, то перше число – максимум*/

max(X, Y, Y) :- X < Y. /*якщо перше число менше другого, то друге число – максимум*/

max(X, Y, Y) :- X = Y. /*якщо перше і друге числа дорівнюють одне одному, то візьмемо за максимум друге*/

Виклик

?- max(51, 38, M).

M=51.

Приклад 2. Встановити взаємозв'язок між назвою книги та її автором за допомогою предиката book(name, author). Програма повинна мати зовнішню мету. У розділ тверджень ввести не менше десяти фактів.

Розв'язок**Вводимо десять фактів**

```
book('Belaya gvardia','Bulgakov').
book('Master i Margarita','Bulgakov').
book('Harry Potter','Rowling').
book('The Casual Vacancy','Rowling').
book('Fahrenheit 451','Bradbury').
book('The Martian Chronicles','Bradbury').
book('The Little Prince','Saint-Exupery').
book('Night Flight','Saint-Exupery').
book('The Hobbit','Tolkien').
book('Lord of the Rings','Tolkien').
```

Створюємо зовнішню мету (функцію обробки фактів) визначення назв книг за автором.

```
book_author(X):-book(Y,X),write(Y).
book_is:-write('Author? '),nl,
        read(X),
        book_author(X),nl,fail.
```

Виклик

```
?- book_is.
Author?
|: 'Rowling'.
Harry Potter
The Casual Vacancy
false.
```

Приклад 3. Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} 1 - \cos(x^3), & \text{if } x < -2 \\ \cos(x^4), & \text{if } -2 < x \leq 2 \\ 10 \sin(x), & \text{if } x > 2 \end{cases}$$

Розв'язок**Задаємо початкове значення для функції**

```
fY(0,6).
```

Створюємо основну функцію для обчислення

```
fY(X,Y):- (X<(-2), Y is 1-cos(X**3));
(X>(-2), X<=2, Y is cos(X**4)+5);
(X>2, Y is 10*sin(X)).
```

Створюємо функцію виклику, введення вхідного значення X та отримання результату Y . У SWI-Prolog, як і будь-якій іншій мові програмування, є зарезервовані функції до читання інформації, яка вводиться з клавіатури, та для виводу: `read()` та `write()` відповідно. Їх будемо використовувати для введення значення змінної та виводу результату розрахунку значення математичної функції.

```
funcY:-write('Enter X'),nl,
        read(X),
        fY(X,Y),
        write('Y(X)= '),write(Y).
```

Виклик функції

?- funcY.

Enter X

|: 3.

Y(X)=1.4112000805986722

true.

?- funcY.

Enter X

|: -1.5.

Y(X)=5.3430020941392815

true .

Завдання для практичної роботи

1. Розробити Пролог-програму встановлення зв'язку між даними (задача № 1).
2. Розробити Пролог-програму реалізації розгалуженого обчислювального процесу (задача № 2).
3. Скласти звіт з роботи, в який додати лістинг програми та копію вікна з результатом виклику створеного предиката.

Варіанти задач*Варіант 1*

1) Встановити взаємозв'язок між ім'ям людини і його професією за допомогою предиката `special (name, profession)`. Програма повинна мати зовнішню мету. У розділ тверджень ввести не менше 10 фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} x^2, & \text{if } x > 0 \\ \sin(x), & \text{if } x < 0 \\ 1, & \text{if } x = 0 \end{cases}$$

Варіант 2

1) Встановити взаємозв'язок між назвою міста, його населенням та площею за допомогою предиката `city (name, people, square)`. Програма повинна мати зовнішню мету. У розділ тверджень ввести не менше 10 фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} \sin^2 x, & \text{if } x > 1 \\ \cos(x^2), & \text{if } x \leq 1 \end{cases}$$

Варіант 3

1) Обчислити суму двох дійсних чисел. Програма повинна мати внутрішню мету, що включає введення з клавіатури доданків і виведення.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} \sin\left(\frac{x^2 + 1}{2}\right), & \text{if } x > 1 \\ \cos(x + 0,5), & \text{if } x \leq 1 \end{cases}$$

Варіант 4

1) Обчислити добуток двох дійсних чисел. Програма повинна мати внутрішню мету, що включає введення з клавіатури множників і виведення.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} \lg(x), & \text{if } x \leq 1 \\ \ln(x), & \text{if } x > 1 \end{cases}$$

Варіант 5

1) Встановити взаємозв'язок між назвою тварини, її середнім зростом та вагою за допомогою предиката `animal (name, height, weight)`. Програма повинна мати зовнішню мету. У розділ тверджень ввести не менше 10 фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} \ln|x|, & \text{if } x \leq 0 \\ \exp(-0.5x^2), & \text{if } x > 0 \end{cases}$$

Варіант 6

1) Встановити взаємозв'язок між назвою фільму, роком його випуску та жанром за допомогою предиката `film (title, year, type)`. Програма повинна мати зовнішню мету. У розділ тверджень ввести не менше 10 фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} x^2 - 1, & \text{if } x \leq 0 \\ 3x + 1, & \text{if } x > 0 \end{cases}$$

Варіант 7

1) Встановити взаємозв'язок між назвою картини, роком її створення та автором за допомогою предиката `picture (title, year, author)`. Програма повинна мати зовнішню мету. У розділ тверджень ввести не менше 10 фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} 3x^3 - 2, & \text{if } x > 1 \\ 3x^3 + 3, & \text{if } x \leq 1 \end{cases}$$

Варіант 8

1) Встановити взаємозв'язки родинних відносин з допомогою предиката `family (parent, child)`. Програма повинна мати внутрішню мету, що дозволяє

вводити з клавіатури ім'я дитини і виводити імена батьків. У розділ тверджень ввести не менше п'яти фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} 2 - \sqrt{1-x}, & \text{if } -1 \leq x \leq 1 \\ 2x^3 - 1, & \text{if } x < -1 \end{cases}$$

Варіант 9

1) Встановити взаємозв'язки родинних відносин за допомогою предикатів `family (people, people)` про наявність безпосереднього родинного зв'язку, `man (people)` про людину-чоловіка, `women (people)` про людину-жінку. Програма повинна містити визначення предиката батьківства `father (people, people)` через вказані вище предикати і внутрішню мету. У розділ тверджень ввести не менше шести фактів по предикатах `man` і `woman`.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} x^2 + 5, & \text{if } x < 0 \\ \sqrt[3]{x^5} + 5, & \text{if } x \geq 0 \end{cases}$$

Варіант 10

1) Ввести координати початку і кінця відрізка в тримірному просторі, обчислити і вивести довжину відрізка.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} \sqrt[3]{x^2} + \exp(x), & \text{if } x < 0 \\ \ln^2(1+x), & \text{if } x \geq 0 \end{cases}$$

Варіант 11

1) Встановити взаємозв'язок між маркою машини, її максимальною швидкістю та вартістю за допомогою предиката `car (type, speed, cost)`. Програма повинна містити внутрішню мету, що дозволяє вивести назви всіх машин з максимальною швидкістю більше 200 км/год., а також мету, яка дозволяє визначити всі машини, дорожчі за вказану ціну. У розділ тверджень ввести не менше десяти фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатур.

$$y = \begin{cases} x^2 + 1, & \text{if } x < 0 \\ |\sin(1-x)|, & \text{if } 1 \leq x \leq 3 \end{cases}$$

Варіант 12

1) Встановити взаємозв'язки між ім'ям дитини і його віком за допомогою предиката `child (name, age)`. Програма повинна містити предикат визначення старшинства за віком `order (name, name)` і зовнішню мету. У розділ тверджень ввести не менше десяти фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} \sin^2 x, & \text{if } x < 0 \\ \sqrt[3]{|\sin(x)|}, & \text{if } x \geq 0 \end{cases}$$

Варіант 13

1) Встановити взаємозв'язки родинних відносин з допомогою предиката `family (people, people)`, що визначає наявність безпосереднього родинного зв'язку. Програма повинна містити визначення предиката `grandfamily (people, people)` з використанням предиката `family` і зовнішню мету. У розділ тверджень ввести не менше десяти фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} \frac{\sin(x)}{x}, & \text{if } x \neq 0 \\ 1, & \text{if } x = 0 \end{cases}$$

Варіант 14

1) Встановити взаємозв'язки родинних відносин з допомогою предиката `parent (people, people)`, що визначає наявність безпосереднього родинного зв'язку. Програма повинна містити визначення предиката `uncle (people, people)` з використанням предиката `parent` і зовнішню мету. У розділ тверджень ввести не менше десяти фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} \frac{2x^3 + 4}{3}, & \text{if } x \geq 0 \\ x^3, & \text{if } x < 0 \end{cases}$$

Варіант 15

1) Встановити взаємозв'язки між ім'ям людини і його номером телефону за допомогою предиката `phone_number (name, number)`. Програма повинна мати внутрішню мету, що включає введення з клавіатури імені, його висновок і виведення телефону. У розділ тверджень ввести не менше шести фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} \cos(x^2 + 1), & \text{if } x < 0 \\ \sin(x^2 - 1), & \text{if } x \geq 0 \end{cases}$$

Варіант 16

1) Встановити взаємозв'язок між назвою музичного твору, жанром та автором за допомогою предиката `music (title, type, author)`. Програма повинна мати зовнішню мету. У розділ тверджень ввести не менше 10 фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} 2|x| - 5, & \text{if } x < 0 \\ \sqrt{1 - x^2}, & \text{if } 0 \leq x \leq 1 \end{cases}$$

Варіант 17

1) Встановити, чи є число, що аналізується, додатнім, нульовим або від'ємним, за допомогою предиката `classify (number, sign)`. Програма повинна мати зовнішню мету.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} x \cos(x), & \text{if } x < -2 \\ \cos(x^2), & \text{if } -2 < x \leq 2 \\ \sin x, & \text{if } x > 2 \end{cases}$$

Варіант 18

1) Встановити взаємозв'язки родинних відносин за допомогою предиката `parent (people, people)`, що визначає наявність безпосереднього родинного зв'язку. Програма повинна містити визначення предиката рідний брат або сестра і зовнішню мету. У розділ тверджень ввести не менше десяти фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} x^2 + 5, & \text{if } x < 0 \\ \sqrt[3]{x^5} + 5, & \text{if } x \geq 0 \end{cases}$$

Варіант 19

1) Встановити взаємозв'язки родинних відносин за допомогою предиката `parent (people, people)`, що визначає наявність безпосереднього родинного зв'язку. Програма повинна містити визначення предиката двоюрідний брат або сестра `cousin (people, people)` через предикат `parent` і зовнішню мету. У розділ тверджень ввести не менше десяти фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} \sqrt[3]{x^2} + \exp(x), & \text{if } x < 0 \\ \ln^2(1 + x), & \text{if } x \geq 0 \end{cases}$$

Варіант 20

1) Встановити взаємозв'язок між назвою витвору мистецтва, роком її створення та автором за допомогою предиката `art (title, year, author)`. Програма повинна мати зовнішню мету. У розділ тверджень ввести не менше 10 фактів.

2) Скласти Пролог-програму обчислення функції $y=f(x)$ для значень аргументу x , що вводиться з клавіатури.

$$y = \begin{cases} x^2 \sin x, & \text{if } x \leq -4 \\ 2 - \cos x^4, & \text{if } -4 < x \leq 2 \\ 4 + \sin^3 x, & \text{if } x > 2 \end{cases}$$

Контрольні питання

1. Поняття терму.
2. Поняття атому, змінної, складного терму.
3. Факти і предикати.
4. Запити у SWI-Prolog.
5. Правила та їх структура і синтаксис.
6. База даних і база знань у SWI-Prolog.
7. Вбудовані функції читання та виводу.

Практична робота № 3

Рекурсія в SWI-Prolog



Мета роботи: отримати навички у використанні рекурсивних викликів предикатів для розв'язання задач мовою програмування SWI-Prolog.

Теоретичні відомості

У SWI-Prolog циклічні обчислення виконуються за допомогою рекурсії. *Рекурсія* – це метод, який часто використовується для досягнення ефекту, аналогічного тому, що реалізується при використанні ітеративних керуючих конструкцій у процедурних мовах. Метод рекурсії має базис і крок.

Базис рекурсії – це речення, яке визначає деяку початкову ситуацію або ситуацію у момент завершення. Як правило, у цьому реченні записується деякий найпростіший випадок, за якого відповідь отримується відразу без використання рекурсії.

Крок рекурсії – це правило, у якому міститься як підциль виклик предиката, який визначається. Для того щоб запобігти зациклюванню, предикат, який визначається, повинен мати виклик не від тих параметрів, які вказані у заголовку правила. Параметри повинні змінюватися на кожному кроці так, щоб в результаті або був виконаний базис рекурсії, або умова виходу з рекурсії, яка розміщується у самому правилі. У загальному вигляді правило, яке реалізує крок рекурсії, виглядає так:

```
<ім'я предикату, який визначається>:-  
    [<підцилі>],  
    [<умова виходу з рекурсії>],  
    [<підцилі>],  
    <ім'я предикату, який визначається>,  
    [<підцилі>].
```

Один зі способів переходу до рекурсивних правил полягає в узагальненні нерекурсивних правил. Наприклад:

```
parent('Anna','Tom').  
parent('Mark','Tom').  
parent('Paul','Monika').  
%предикат для визначення предків  
ancestor(Ancessor, Descendant):- parent(Ancessor, Descendant).  
ancestor(Ancessor, Descendant):- parent(Ancessor,X),  
                                   ancestor(X,Descendant).
```

Запит матиме вигляд:

```
?- ancestor(X,'Tom').  
X='Anna'.  
X='Mark'.
```

Рекурсивний опис правила містить у собі посилання на заголовок цього правила.

Обчислювальна рекурсія. Розглянемо варіант розв'язання задачі визначення факторіалу числа n .

```
factorial(0,1):-!. /*умова виходу з рекурсії*/  
factorial(N,F):-  
    N1 is N-1,  
    factorial(N1,F1),  
    F is N*F1.
```

У розв'язку спочатку вказана умова виходу з рекурсії. Далі записується рекурсивне правило `factorial`, аргументами якого є число N , для якого обчислюється факторіал, та F – результат обчислення. У тілі правила записується обчислювальне рекурсивне визначення факторіалу.

У наведеному визначенні факторіалу використовується механізм *відсікання* (`cut`) «!». Відсікання є вбудованим предикатом і застосовується у випадках, коли потрібно змінити процес пошуку рішень.

Виділяють три випадки використання відсікання:

1) якщо за деяких умов будь-яка ціль ніколи не повинна бути успішною. Комбінація `cut-fail` виключить виконання інших правил, які відповідають даній цілі.

Наприклад, предикат `not(P)` можна визначити за допомогою відсікання:

```
not(P):-P,!,fail.  
not(_):-true.
```

2) для запобігання нескінченних циклів. У наведеному прикладі для визначення факторіалу за допомогою відсікання забезпечується вихід з рекурсії. При виконанні першого правила за предикатом відсікання відбувається «заморожування» всіх альтернативних тверджень для `factorial`, які стоять нижче першого правила, тобто припиняється виконання рекурсивного правила;

3) при програмуванні взаємовиключних тверджень. Наприклад,

```
sign(X,-1):-X<0,!.  
sign(X,0):-X=0,!.  
sign(X,1):-X>0.
```

Вивід з останнього правила станеться лише тоді, коли $X>0$, у всіх інших випадках це правило не буде навіть розглядатися як альтернатива у точці повернення.

Крім відсікання, для організації циклічних правил використовується предикат *відмови* `fail`. Предикат `fail` застосовується для отримання гарантованого неуспіху при доведенні деякої цілі. Наприклад, правило `A:-B,fail.` буде виконуватися стільки разів, скільки є альтернатив для B у ньому.

Розглянемо приклад використання предикату відмови. Створимо предикат, який буде виводити всі імена студентів.

```
student('Math','Mary'). %першим аргументом є назва факультету,  
другим – ім'я студента  
student('Math','Loise').  
student('Math','Fred').  
student('Socio','Hanna').  
student('Socio','Eddie').  
student('Computer','Annete').
```

```

student('Computer','Karl').
stud_names:-student(_,Name), %виводить імена студентів з усіх
факультетів
    write(Name),nl,
    fail.
stud_facult(X):-student(X,Name), %виводить імена студентів з
одного факультету
    write(Name),nl,
    fail.

```

Приклад виклику предикатів

```

?- stud_names.
Mary
Loise
Fred
Hanna
Eddie
Annete
Karl
?- stud_facult('Math').
Mary
Loise
Fred

```

Розглянемо декілька прикладів використання обчислювальної рекурсії.

Приклад 1. Предикат, що визначає n-не число Фібоначчі. Як перші два числа взяти 1.

```

fibonachi(1,1). %перше число ряду - 1
fibonachi(2,1). %друге число ряду - 1
fibonachi(N,F):-N>2, %визначення чисел, починаючи з третього
    N1 is N-1,
    fibonachi(N1,F1),
    N2 is N-2,
    fibonachi(N2,F2),
    F is F1+F2.

```

Виклик предиката

```

?- fibonachi(6,X).
X = 8 .

```

Приклад 2. Визначення суми непарних чисел з перших n чисел Фібоначчі.

```

odd(X):-P is (X mod 2), P==1. %визначення непарного числа
sumfibodd(N,X):-sumfibodd(N,0,X). %на початку сума дорівнює 0
sumfibodd(N,Sm,X):-N>0,
    N1 is N-1,
    fibonachi(N1,F),
    (odd(F)-> Sm1 is Sm+F; Sm1 is Sm),
    sumfibodd(N1,Sm1,X).
sumfibodd(0,Sm1,Sm1). %якщо кількість чисел 0, то сума не
змінюється

```

Виклик предиката

```

?- sumfibodd(7,X).
X = 23

```

Приклад 3. Обчислення функції Аккермана, визначеної на множині пар невід'ємних чисел

$$A(X, Y) = \begin{cases} Y + 1, \text{ якщо } X = 0; \\ A(X, -1, 1), \text{ якщо } X > 0, Y = 0; \\ A(X - 1, A(X, Y - 1)), \text{ якщо } X > 0, Y > 0. \end{cases}$$

```

akkerman(0,0,1).
akkerman(0,1,2).
akkerman(X,Y,Z):-
    (X==0,Z is Y+1);
    (X>0, Y==0, X1 is X-1, akkerman(X1,1,Z));
    (X>0,Y>0,X1 is X-1, Y1 is Y-1, akkerman(X,Y1,R1),
akkerman(X1,R1,Z)).

```

Виклик предиката

```

?- akkerman(3,2,X).
X = 29

```

Приклад 4. Обчислити суму 10 членів функціонального ряду $\sum_{i=1}^{10} \frac{x^{2i-1}}{i!}$ для заданого значення $x=0.5$.

```

factorial(1,1). %визначення факторіалу
factorial(N,F):- N>1,
    N1 is N-1,
    factorial(N1,F1),
    F is N*F1.
sum(_,0,Y):-Y is 0. %початкове значення суми
sum(X,N,Y):-
    N1 is N-1,
    factorial(N,F),
    sum(X,N1,Y1),
    Y is Y1+(X^(2*N-1)/F).

```

Виклик предиката

```

?- sum(0.5,10,S).
S = 0.5680508333754707

```

Завдання для практичної роботи

1. Розробити програму мовою SWI-Prolog для обчислення суми членів функціонального ряду для заданої кількості його членів n (з використанням рекурсії). Значення аргументу x взяти з вказаного діапазону.

2. Скласти звіт з роботи, в який додати лістинг програми та копію вікна з результатом виклику створених предикатів.

Варіанти завдань

№	Функціональний ряд	Діапазон зміни аргументу	Кількість членів ряду n
1	2	3	4
1	$\sum_{i=0}^n \frac{\ln^i 3}{i!} x^i$	[0.1;1]	10
2	$\sum_{i=1}^n \frac{\cos(ix)}{i}$	$[\pi/5; 9\pi/5]$	16
3	$\sum_{i=0}^n (-1)^i \frac{x^{2i+1}}{(2i+1)}$	[0.1;1]	10
4	$\sum_{i=1}^n (-1)^{i+1} \frac{\sin(ix)}{i}$	$[\pi/5; 4\pi/5]$	18
5	$\sum_{i=0}^n \frac{x^i}{i!}$	[1;2]	15
6	$\sum_{i=0}^n \frac{\cos(i\pi/4)}{i} x^i$	[0.1;1]	13
7	$\sum_{i=0}^n (-1)^i \frac{x^{2i}}{(2i)!}$	[0.1;1]	10
8	$\sum_{i=1}^n x^i \sin(i\pi/4)$	[0.1;0.8]	14
9	$\sum_{i=0}^n \frac{x^{4i+1}}{(4i+1)}$	[0.1;0.8]	17
10	$\sum_{i=0}^n \frac{\cos(x)}{i!}$	[0.1;1]	20
11	$\sum_{i=0}^n \frac{x^{2i}(2i+1)}{i!}$	[0.1;1]	10
12	$\sum_{i=1}^n \frac{\cos(i\pi/3)}{i} x^i$	[0.1;0.8]	18
13	$\sum_{i=1}^n \frac{1}{(2i+1)} \left(\frac{x-1}{x+1}\right)^{2i+1}$	[0.2;1]	11

1	2	3	4
14	$\sum_{i=1}^n (-1)^i \frac{\cos(ix)}{i^2}$	$[\pi/5; \pi]$	20
15	$\sum_{i=1}^n (-1)^{i+1} \frac{x^{2i+1}}{(4i^2 - 1)}$	$[0.1; 1]$	15
16	$\sum_{i=1}^n \frac{\sin((2i-1)x)}{2i-1}$	$[\pi/10; 9\pi/10]$	14
17	$\sum_{i=0}^n \frac{x^{2i}}{(2i)!}$	$[0.1; 1]$	12
18	$\sum_{i=1}^n \frac{\cos(2ix)}{4i^2 - 1}$	$[0.1; 0.8]$	13
19	$\sum_{i=0}^n \frac{(2x)^i}{i!}$	$[0.1; 1]$	20
20	$\sum_{i=0}^n \frac{i^2 + 1}{i!} \left(\frac{x}{2}\right)^i$	$[0.1; 1]$	17

Контрольні питання

1. Поняття рекурсії.
2. Базис та крок рекурсії.
3. Відсікання у SWI-Prolog.
4. Відмова у SWI-Prolog.
5. Особливості обчислювальної рекурсії.

Практична робота № 4

Списки у SWI-Prolog



Мета роботи: ознайомитись з поняттям списку у SWI-Prolog та навчитись використовувати списки для розв'язання задач мовою SWI-Prolog.

Теоретичні відомості

Список – це упорядкована послідовність елементів. Елементами списку можуть бути будь-які терми SWI-Prolog. Зручною формою запису списків є списочне позначення. У цьому позначенні кожний елемент списку відокремлюється від іншого комою, а весь набір елементів береться у квадратні дужки, наприклад [a,b,c,d].

Фактично список – це структура з функтором ‘.’/2 (точка з арністю 2). За цим визначенням список складається з першого елемента і хвоста, який є списком з решти елементів. *Пустий* список – це список, який не містить жодного елемента, він позначається []. Приклади списків наведені у табл. 4.

Таблиця 4

Приклади списків та їх записів

№	Елементи списку	Запис із дужками	Запис з функтором ‘.’
1	a	[a]	‘.’(a,[])
2	a b c	[a,b,c]	‘.’(a, ‘.’(b, ‘.’(c,[])))
3	[a]	[[a]]	‘.’(‘.’(a,[]),[])
4	[]	[[]]	‘.’([],[])
5	[a] [b, c]	[[a],[b,c]]	‘.’(‘.’(a,[]), ‘.’(b, ‘.’(c,[])))

Список уніфікується з іншим списком, якщо попарно уніфікуються їх елементи.

Список можна ділити на «голову» і «хвіст» за допомогою операції відділення голови, яка позначається вертикальною рисою (|), тобто [Голова|Хвіст]. *Голова* – фіксована кількість елементів. *Хвіст* – список із решти елементів списку. Найчастіше використовується голова, яка складається з одного елемента. Приклади списків наведені у табл. 5.

Таблиця 5

Приклади співставлення списку зі списком [Голова|Хвіст]

№	Список	Голова	Хвіст
1	[a]	A	[]
2	[a,b,c]	A	[b,c]
3	[[a]]	[a]	[]
4	[]	не співставляється	не співставляється
5	[a [c,d]]	A	[c,d]

Список $[1,2|[3,4]]$ дорівнює списку $[1,2,3,4]$. При співставленні зі списком $[X,Y|Z]$ отримаємо $X=1$, $Y=2$ і $Z=[3,4]$.

У SWI-Prolog є особливий вид списків – символні списки. Символьний список – це фактично послідовність цілих чисел, які відповідають ASCII-коду символів. Символьні списки беруться у подвійні лапки.

Приклад списків, які можна співставити: “abc”, $[97,98,99]$, ‘.’(97, ‘.’(98, ‘.’(99,[]))).

Розглянемо декілька прикладів роботи зі списками.

Приклад 1. Поділити список на голову і хвіст.

```
write_list([]).
write_list([H|T]):- /* виділити голову і хвіст, */
write(H), nl, /* вивід голови */
write_list(T)./*рекурсивний виклик для решти елементів списку*/
```

Приклад 2. Об’єднання і розділення списків.

```
append([],List,List).
append([X|L1],List2,[X|L3]):-append(L1,List2,L3).
```

Виклик предиката

```
?- append([s,d,f],[b,n,m],X).
X = [s, d, f, b, n, m].
```

Зауваження. Предикат `append/3` є вбудованим у SWI-Prolog.

Приклад 3. Використовуючи поняття списку, напишіть програми для наступних задач:

а) визначення довжини списку (довжина пустого списку дорівнює 0; довжина непустого списку дорівнює довжині його хвоста плюс 1).

```
list_lenght([],0).
list_lenght([_|T],N):-
    list_lenght(T,N1),
    N is N1+1.
?- list_lenght([s,b,g,5,r,4],X).
X = 6.
```

б) визначити, чи належить елемент списку (елемент належить списку, якщо він – його голова; елемент належить списку, якщо він належить хвосту списку).

```
inlist(R,[R|_]).
inlist(R,[_|T]):-
    inlist(R,T).
?- inlist(s,[4,6,7,h,n,b]).
false.
?- inlist(n,[4,6,7,h,n,b]).
True
```

в) пошук суми елементів числового списку.

```
sum_list([],0).
sum_list([H|T],S):-
    sum_list(T,S1),
    S is S1+H.
?- sum_list([3,6,7,2,9],S).
S = 27.
```

г) пошук максимального і мінімального елементів списку.

```
max_list([X],X).
max_list([H|T],H):-max_list(T,M),H>M.
max_list([_|T],M):-max_list(T,M).
min_list([X],X).
min_list([H|T],H):-min_list(T,M),H<M.
min_list([_|T],M):-min_list(T,M).
?- max_list([3,6,7,2,9],M).
M = 9.
```

Завдання для практичної роботи

1. Виконати приклади 1, 2, 3 для списку, заданого самостійно.
2. Розробити програму мовою SWI-Prolog для обробки списку відповідно до варіанта.
3. Скласти звіт з роботи, в який додати лістинг програми та копію вікна з результатом виклику створених предикатів.

Варіанти завдань

1. У списку $D = (d_1, d_2, \dots, d_{12})$ визначити середнє арифметичне негативних елементів S та їх кількість K .
2. У списку $Z = (z_1, z_2, \dots, z_{14})$ визначити добуток P елементів з парними індексами і суму S елементів з непарними індексами.
3. У списку $A = (a_1, a_2, \dots, a_{13})$ визначити суму S квадратів елементів, більших ніж 2.3 по абсолютному значенню, та їх кількість K .
4. У списку $C = (c_1, c_2, \dots, c_{11})$ визначити добуток P елементів більших нуля і середнє арифметичне S елементів менших нуля.
5. У списку $T = (t_1, t_2, \dots, t_{12})$ визначити добуток P позитивних елементів, кількість K нульових елементів та суму S негативних елементів.
6. Для списку $F = (f_1, f_2, \dots, f_{13})$ визначити число D , яке дорівнює різниці суми S позитивних елементів і SR середнього арифметичного негативних елементів.
7. У списку $T = (t_1, t_2, \dots, t_{14})$ визначити добуток P відмінних від 0 елементів з парними номерами та їх кількість K і середнє арифметичне S відмінних від 0 елементів з непарними номерами.
8. У списку $Q = (q_1, q_2, \dots, q_{15})$ визначити суму S і добуток P елементів, більших за 0. Якщо сума більша за добуток, то позитивні елементи замінити нулями; в іншому випадку замінити на -1.
9. У списку $Q = (q_1, q_2, \dots, q_{13})$ визначити найближчий нульовий елемент і всі наступні елементи замінити нулями.
10. У списку $C = (c_1, c_2, \dots, c_{16})$ визначити середнє арифметичне SR негативних елементів першої половини списку та добуток P позитивних елементів другої половини списку.
11. У списку $B = (b_1, b_2, \dots, b_{10})$ визначити M максимальний елемент та його індекс. Максимальний елемент замінити на 0.

12. У списку $A = (a_1, a_2, \dots, a_{14})$ визначити, при якому K кількість елементів списку та їх сума S перебільшить певне число Q .

13. У списку $Y = (y_1, y_2, \dots, y_{13})$ визначити мінімальний і максимальний елементи списку та їх індекси.

14. У списку $X = (x_1, x_2, \dots, x_{10})$ визначити мінімальний елемент і поставити його на перше місце у списку.

15. У списку $C = (c_1, c_2, \dots, c_{10})$ визначити максимальний елемент і поставити його на останнє місце у списку.

16. У списку $B = (b_1, b_2, \dots, b_{10})$ максимальний і мінімальний елементи поміняти місцями.

17. Вивести значення елементів списку $X = (x_1, x_2, \dots, x_{12})$ та їх номери, які задовольняють умові $a < x_i < b$, де a та b – певні числа.

18. У списку $Y = (y_1, y_2, \dots, y_{13})$ визначити \min мінімальний елемент списку. Замінити нульові елементи на \min і вивести їх номери.

19. У списку $A = (a_1, a_2, \dots, a_{14})$ визначити суму S перших трьох і добуток P останніх трьох елементів списку.

20. У списку $H = (h_1, h_2, \dots, h_{14})$ поміняти місцями елементи з парними і непарними номерами.

Контрольні питання

1. Визначення списку.
2. Форми запису списку.
3. Голова і хвіст списку.
4. Символьні списки.
5. Які списки уніфікуються?
6. Як співставляти списки?

Практична робота № 5

Робота з базами даних у SWI-Prolog



Мета роботи: отримати навички у редагуванні та обробці баз даних мовою програмування SWI-Prolog.

Теоретичні відомості

Набір фактів у SWI-Prolog є базою даних. Базы даних можуть бути статичними (без змін) або *динамічними* (яка може змінюватись у процесі роботи з нею). У SWI-Prolog є можливість створити динамічну базу даних. Для цього необхідно вказати предикат як змінну:

```
:- dynamic <ім'я предикату>/<арність>.
```

Розглянемо наприклад базу даних щодо членів студентського клубу, яка містить інформацію про: прізвище, вік чи сплачений внесок.

```
:- dynamic member/3.  
member('Vakulenko', 33, pay).  
member('Ivanenko', 15, pay).  
member('Vovchenko', 27, pay).  
member('Petrenko', 20, no_pay).  
member('Khomenko', 25, no_pay).
```

Для *додавання* фактів до бази даних використовують вбудовані предикати `assert` (у кінець) і `asserta` (у початок). Наприклад:

```
assert(member(Surname, Age, Pay_data)).
```

Для *видалення* фактів з бази даних є вбудовані предикати `retract` (видалити лише один запис) і `retractall` (видалити усі записи для заданого значення аргументу). Наприклад:

```
%видалення всіх записів із вказаним прізвищем  
retractall(member(Surname, _, _)).
```

Для того щоб змінити запис у базі даних, необхідно його спочатку видалити, а потім замість нього додати інший з новою інформацією.

Розглянемо роботу з базою даних про членів клубу. Передбачимо такі можливості: перегляд інформації, додавання нового запису, видалення запису і зміна запису.

Для початку необхідно видалити будь-яку інформацію про розглядувану базу даних, якщо вона зберігається у пам'яті, і завантажити базу даних із файлу, де вона зберігається:

```
start:-  
    retractall(member/3),  
    reconsult('club_members.pl'),
```

```
dynamic(member/3),
menu.
```

В останньому рядку йде виклик запиту menu, у якому описані дії для роботи з базою даних. Запит reconsult() завантажує на виконання файл, ім'я якого вказане у дужках. У предикаті dynamic() вказується ім'я бази даних.

Далі записується запит menu:

```
menu:-repeat,
    nl,nl,write("Menu"),nl,
    write("1. Information about club members"),nl,
    write("2. Change information about club members"),nl,
    write("3. Add club member record"),nl,
    write("4. Delete club member record"),nl,
    write("0. Output"),nl,nl,
    write("Enter the menu item number: "),
    read(X),
    process(X),
    db_save.
```

Тут користувачу надається перелік дій з базою даних, і програма чекає поки буде введений номер дії (read(X)), потім відповідно до введеного номера викликається дія (process(X)) і в кінці при завершенні роботи всі зміни у базі даних зберігаються (db_save).

Якщо користувач вводить «0» виконання програми завершується, якщо вводить номер від 1 до 4, то здійснюється перехід до подальших дій.

```
process(0):-!.
process(X):-item(X),fail.
```

Якщо введено «1», то користувачу надається поточна інформація з бази даних.

```
item(1):-nl,write("Club members"),nl,
    write("Surname\t\t"),write("Age\t"),write("Summ\t"),
    nl,give_info(_),!.
```

Запит give_info() з бази даних зчитує інформацію та виводить користувачу.

```
give_info(member(Surname, Age, Pay_data)):-
    member(Surname, Age, Pay_data),
    write(Surname),write("\t"),write(Age),write("\t"),
    write(Summ),write("\t"),write(Pay_data),
    nl,fail,
    give_info(_).
```

Якщо введено «2», то користувачу надається можливість змінити деяку інформацію у базі даних. Наприклад, змінити прізвище.

```
item(2):-nl,change_member,
    write("The information was changed."),
    nl,!.
```

```
change_member:-
    nl,write("Change information about club member: "),nl,
```

```

write("of Surname/Enter 1\n"),write("of Age/Enter 2\n"),
write("of Payment/Enter 3"),read(N),
change(N),
db_save,
nl.
change(1):-write("Enter member's Surname "),read(Surname),nl,
write("Enter new Surname "),read(New_Surname),
member(Surname, Age, Pay),
A=Age,
P=Pay,
retract(member(Surname,_,_)),
asserta(member(New_Surname,A,P)).

```

Змінювати інформацію можна за будь-яким параметром, наприклад, якщо внесок був сплачений, то необхідно змінити відповідний статус.

Якщо введено «3», то користувач має можливість додати новий запис.

```

item(3):-nl,write("Enter      information      about      new      club
member"),nl,
write("Surname "),read(Surname),
write("Age"),read(Age),
write("Contribution payment stamp "),read(Pay_data),
add_member(member(Surname, Age, Pay_data)),!.
add_member(Member):-assertz(Member).

```

Якщо введено «4», то користувач має можливість видалити запис за деяким параметром (наприклад, без сплати внеску при N=3).

```

item(4):-nl,write("Delete information about club member"),nl,
write("of Surname/Enter 1\n"),write("of Age/Enter 2\n"),
write("of Payment/Enter 3"),read(N),
((N=1,write("Enter      Surname"),read(Surname),
delete_member(member(Surname,_,_)));
(N=2,write("Enter
Age"),read(Age),delete_member(member(_,Age,_)));
(N=3,write("Delete      all      without      payment"),
delete_member(member(_,_,no_pay))),!.
delete_member(Member):-retractall(Member).

```


Для збереження оновленої бази даних у файл передбачений предикат db_save.

```

db_save:-
tell('club_members.txt'), %відкрити файл для запису
listing(member), %записати дані про членів клубу у файл
told. %закрити файл

```

На рис. 1–2 показаний приклад запуску і роботи з наведеним меню. На рис. 1 показаний початок роботи з програмою і виведення користувачу інформації з бази даних.



```
SWI-Prolog (AMD64, Multi-threaded, version 8.0.3)
File Edit Settings Run Debug Help
?- consult(club).
true.
?- start.

Menu
1. Information about club members
2. Change information about club members
3. Add club member record
4. Delete club member record
0. Output

Enter the menu item number: 1.

Club members
Surname      Age      Summ      Payment
Vakulenko    33      grn(2)    pay
Khomenko     25      grn(2)    pay
Ivanenko     15      grn(1)    pay
Vovchenko    27      grn(2)    pay
Ivanenko     16      grn(1)    pay
Petrenko     20      grn(2)    no_pay
Khomenko     25      grn(2)    no_pay
Gorobchenko  20      grn(2)    pay
Tkachenko    17      grn(1)    pay
Udovenko     17      grn(1)    pay
Ivchenko     23      grn(2)    no_pay
```

*Рис. 1 – Виведення інформації
про членів студентського клубу*

На рис. 2 показано, як видалені з бази даних записи з прізвищем «Ivanenko».



```
SWI-Prolog (AMD64, Multi-threaded, version 8.0.3)
File Edit Settings Run Debug Help

Menu
1. Information about club members
2. Change information about club members
3. Add club member record
4. Delete club member record
0. Output

Enter the menu item number: |: 4.

Delete information about club member
of Surname/Enter 1
of Age/Enter 2
of Payment/Enter 3|: 1.
Enter Surname|: 'Ivanenko'.
```

Рис. 2 – Видалення записів за прізвищем

Після видалення можна знов вивести оновлену інформацію для перевірки роботи програми (рис. 3).



Рис. 3 – Перевірка видалення записів і вихід з меню

Таким чином, створене достатньо просте і зрозуміле меню для роботи з простою базою даних. Наведений приклад бази даних є показовим, оскільки не містить ключові елементи, які б ідентифікували конкретного члена клубу.

Завдання для практичної роботи

1. Розробити програму для роботи з базою даних за варіантом.
2. Створити меню для обробки бази даних. Обов'язково мають бути запити: додавання нового рядка, зміна запису та видалення запису з бази даних. Кількість фактів у базі даних повинна бути 10 і більше.
3. Скласти звіт з роботи, в який додати лістинг програми та копію вікна з результатом виклику пунктів меню.

Варіанти тем баз даних:

1. Конфігурація персонального комп'ютера.
2. Програмне забезпечення для персонального комп'ютера користувача.
3. Тарифні плани операторів мобільного зв'язку.
4. Обладнання для комп'ютерної мережі.
5. Програмне забезпечення для підприємства.
6. Пакети прикладних програм для розв'язання задач.
7. Програмне забезпечення для комп'ютерної мережі.
8. Наукова література у бібліотеці (для написання реферату тощо).
9. Спеціальності для навчання у вищому навчальному закладі.
10. Тури для відпочинку або подорожі.
11. Поточна успішність студентів на факультеті.
12. Результати сесії на факультеті.
13. Розклад занять у семестрі.
14. Інформація про пацієнта у лікарні.

15. Забезпеченість літературою навчального процесу.
16. Інформація про співробітників на підприємстві (декілька відділів).
17. Розклад руху поїздів на станції.
18. Художня література у бібліотеці (для школярів).
19. Інформація про навчальні заклади у регіоні.
20. За вибором студента.

Контрольні питання

1. Особливості бази даних у SWI-Prolog.
2. Як вказати, що база даних є динамічною?
3. Які є предикати для додавання записів?
4. Які є предикати для видалення записів?
5. Як змінити запис у базі даних?
6. Як зберегти зміни у базі даних?

Практична робота № 6

Пошук у просторі станів



Мета роботи: ознайомитися з алгоритмами пошуку у просторі станів; реалізувати алгоритми пошуку у шир, у глибину та жадібний пошук мовою SWI-Prolog.

Теоретичні відомості

Пошук у глибину

Пошук у глибину просто шукає будь-який шлях між двома станами, тому немає ніяких гарантій, що він буде найкоротшим. Цей пошук можна використовувати для пошуку всіх шляхів між двома станами. Застосовується для незважених графів.

Припустимо, що заданий неорієнтований граф (рис. 4). І потрібно знайти всі шляхи від вершини «а» до вершини «с».

Спочатку задаємо набір фактів, які будуть відображати ребра графа, а також визначаємо правило одного кроку move.

```
m(a,b).  
m(b,c).  
m(a,d).  
m(b,d).  
m(c,d).  
m(c,e).  
m(d,e).  
move(A,B):-m(A,B);m(B,A).
```

%оскільки граф неорієнтований

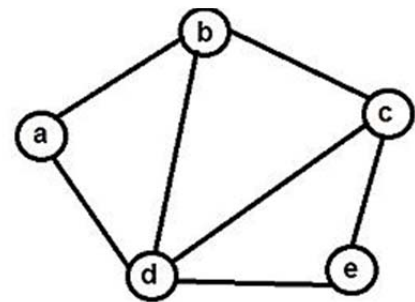


Рис. 4 – Неорієнтований граф

Шлях, який пройшли, зберігаємо у вигляді списку станів, але у зворотному порядку, тобто стартова вершина буде розташовуватися вкінці, а поточна – на початку.

Щоб продовжити шлях на один крок, необхідно знайти, який крок можна зробити, і щоб не було зациклювання – перевірити, щоб цей крок не привів до стану, у якому вже були.

```
prolong([Temp|Tail],[New,Temp|Tail]):-  
    move(Temp,New),not(member(New,[Temp|Tail])).
```

Предикат пошуку у глибину:

```
dpth([Finish|Tail],Finish,[Finish|Tail]). %якщо поточна вершина  
%співпадає з кінцевою, то шлях знайдений  
dpth(TempWay,Finish,Way):-  
    prolong(TempWay,NewWay), %намагаємося зробити крок  
    dpth(NewWay,Finish,Way).%продовжуємо пошук уже з урахуванням  
%зробленого кроку
```

Допоміжний предикат для зручності користувача:

```
search_dpth(Start,Finish):-
    dpth([Start],Finish,Way),%викликаємо пошук у глибину,
    вважаючи, що
    %шлях складається лише з початкової вершини
    show_answer(Way).%виводимо шлях на екран у наочному вигляді
```

Для виводу на екран використовуємо спеціальний предикат, у якому враховується, що шлях зберігається у зворотному порядку:

```
show_answer([_]):-!.
show_answer([A,B|Tail]):-
    show_answer([B|Tail]),nl,write(B),write(' -> '),write(A).
```

Результат роботи методу

```
?- search_dpth(a,c).
```

```
a -> b
b -> c
true
```

Пошук усіх шляхів

```
?- search_dpth(a,c),nl,nl,nl,fail.
```

```
a -> b
b -> c
```

```
a -> b
b -> d
d -> e
e -> c
```

```
a -> b
b -> d
d -> c
```

```
a -> d
d -> e
e -> c
```

```
a -> d
d -> b
b -> c
```

```
a -> d
d -> c
```

```
false.
```

Пошук у ширину

Цей вид пошуку також застосовується для незважених графів, але може знаходити найкоротший шлях (або за бажанням найдовший).

Будемо зберігати не один шлях, який був пройдений, а список усіх можливих шляхів, які можна було б пройти. Вони будуть розташовані від більш коротких до більш довгих. Кожний з них, як і у пошуку у глибину, буде записаний у зворотному порядку.

```
%якщо у поточного шляха перша вершина співпадає з кінцевою,
%то даний шлях є відповіддю
bdth([[Finish|Tail]|_],Finish,[Finish|Tail]).
```

```

bdth([TempWay|OtherWays],Finish,Way):-
    findall(W,prolong(TempWay,W),Ways),%також можна знайти всі
    %способи, за якими можна зробити крок з поточного стану
першого шляху
    append(OtherWays,Ways,NewWays),%додаємо всі ці способи у
кінець
    %списку шляхів
    bdth(NewWays,Finish,Way).%і продовжуємо пошук

```

Оскільки всі шляхи у списку розташовуються від більш коротких до більш довгих (хоча їхня довжина найчастіше відмінна лише на один крок, оскільки після огляду шлях відразу відкидається), то коли вперше виконується правило `bdth([Finish|Tail]|_, Finish, [Finish|Tail])`, знайдений шлях гарантовано буде найкоротшим.

Також потрібно змінити допоміжний предикат

```

search_bdth(Start,Finish):-
    bdth([Start],Finish,Way),%спочатку список шляхів
складається
    %з одного шляху, який містить початкову вершину
    show_answer(Way).

```

Усі шляхи у графі у порядку збільшення довжини

```
?- search_bdth(a,c),nl,nl,nl,fail.
```

```
a -> d
```

```
d -> c
```

```
a -> b
```

```
b -> c
```

```
a -> d
```

```
d -> e
```

```
e -> c
```

```
a -> d
```

```
d -> b
```

```
b -> c
```

```
a -> b
```

```
b -> d
```

```
d -> c
```

```
a -> b
```

```
b -> d
```

```
d -> e
```

```
e -> c
```

```
false.
```

Якщо потрібний навпаки найдовший шлях, то необхідно поміняти місцями `bdth`.

Усі шляхи у графі у порядку зменшення довжини

```
?- search_bdth(a,c),nl,nl,nl,fail.
```

```

a -> b
b -> d
d -> e
e -> c

a -> b
b -> d
d -> c

a -> d
d -> b
b -> c

a -> d
d -> e
e -> c

a -> b
b -> c

a -> d
d -> c

false.

```

Пошук з ітераційним заглибленням

Також створений для незважених графів. Головним недоліком пошуку у ширину є великий об'єм використаної пам'яті, а пошук з ітераційним заглибленням навпаки використовують її економно. Сенс даного пошуку полягає у тому, що перебираємо максимальну глибину пошуку, починаючи з 1 і до якогось обмежувального числа, і для кожної такої максимальної глибини запускається пошук у глибину з відповідним обмеженням. Таким чином спочатку шукають усі шляхи довжини 1, потім усі шляхи довжини 2, усі шляхи довжини 3 і т. д. Як тільки знайдеться розв'язок, цей знайдений шлях і буде найкоротшим.

Спочатку визначимо предикат, який генерує максимальну глибину пошуку, тобто цілі числа від 1 і далі.

```

int(1).
int(N):-int(M),N is M+1.

```

Змінюємо предикат `depth` з пошуку у глибину з урахуванням того, що глибина пошуку буде обмежена.

```

id([Finish|Tail],Finish,[Finish|Tail],0).
id(TempWay,Finish,Way,N):-N>0,
    prolong(TempWay,NewWay),N1 is N-1,
    id(NewWay,Finish,Way,N1).

```

Зрештою створюємо головний предикат, який перебирає обмеження глибини і визиває предикат `id`.

```

search_id(Start,Finish):-
    int(Level),%вибираємо наступне значення обмеження глибини
    (Level>100,!;%обов'язково треба поставити обмеження на неї
    оскільки
    %якщо шлях взагалі не існує, то без перевірки програми
    зациклюється

```

```
id([Start],Finish,Way,Level),%якщо глибина допустима, то
%викликається пошук
show_answer(Way)).
```

Пошук на основі вагової функції

Розглянемо пошук для зважених графів.

Нехай заданий зважений граф (рис. 5).

Задамо довжини ребер.

```
m(a,b,10).
m(b,c,7).
m(a,d,3).
m(b,d,5).
m(c,d,15).
m(c,e,7).
m(d,e,5).
move(A,B,C):-m(A,B,C);m(B,A,C).
```

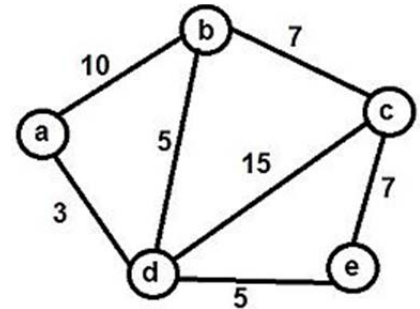


Рис. 5 – Зважений граф

Будемо зберігати не тільки шлях, який був пройдений, але й його довжину у вигляді

Довжина:[ПоточнаВершина, ПопередняВершина ... ПочатковаВершина].

Усі розглянуті шляхи будуть відсортовані у списку не за кількістю вузлів, а за їх довжиною, для цього введемо додаткові предикати.

Предикат, який додає новий шлях у список уже згенерованих шляхів на потрібне місце.

%предикат placeone приймає як параметр новий шлях, а також отримані до

%цього шляхи, які вже відсортовані відносно їх довжини, і повертає

%новий список шляхів, отриманий після додатку першого параметру у %список шляхів на потрібне місце, тобто зберігаючи відсортованість

%якщо довжина нового шляху не більша довжини шляху, який знаходиться

%на першому місці, тоді цей новий шлях розташовуємо на перше місце

```
placeone(Length:Way,[LengthH:WayH|Tail],[Length:Way,LengthH:WayH|Tail]):-Length=<LengthH,!.
```

%з-за відсічення сюди потрапляємо лише якщо 1 правило не виконалось,

%отже перший шлях у списку коротший того, який додається, тому %рекурсивно додаємо його у хвіст

```
placeone(LengthWay,[LengthHWayH|Tail],[LengthHWayH|NewTail]):-
placeone(LengthWay,Tail,NewTail).
```

%окремо розглядається випадок, коли список шляхів пустий

```
placeone(LengthWay,[],[LengthWay]).
```

Предикат, який розставляє шляхи зі списку по потрібним місцям.

%Предикат place приймає як параметри лише щойно згенеровані шляхи

%(які розташовані у випадковому відносно їх довжини порядку)

%до цього згенеровані шляхи (які вже навпаки відсортовані),


```

%i розставляє шляхи з першого списку у другий список на потрібні
місця,
%щоб відсортованість зберігалась
place([],SortedWays,SortedWays).%якщо список пустий, то поточний
список i буде відповіддю
place([Way|Tail],PrevWays,SortedWays):-
    placeone(Way,PrevWays,PrevWays1),%інакше вставляємо на
потрібне місце перший шлях
    place(Tail,PrevWays1,SortedWays).%i продовжуємо розставляти
решту

```

Далі потрібно змінити предикат пошуку, додаючи у нього вищеописаний предикат розстановки.

```

bst([Length:[Finish|Tail]|_],Finish,Length:[Finish|Tail]).
bst([TempWay|OtherWays],Finish,Way):-
    findall(W,prolong(TempWay,W),Ways),
    place(Ways,OtherWays,NewWays),%ось відмінність від пошуку
в ширину
    %нові шляхи не додаються в кінець, а розставляються по
потрібним місцям
    bst(NewWays,Finish,Way).

```

І не забуваємо змінити допоміжний предикат, в якому вважаємо, що початкова довжина шляху дорівнює 0.

```

bst([0:[Start]],Finish,Length:Way),
show_answer(Way),nl,write('Length of way: '),write(Length).

```

Результат роботи програми:

```

?- search_bst(a,c).

```

```

a -> d
d -> e
e -> c
Length of way: 15
true

```

Усі шляхи в порядку зростання довжини

```

?- search_bst(a,c),nl,nl,nl,fail.

```

```

a -> d
d -> e
e -> c
Length of way: 15

```

```

a -> d
d -> b
b -> c
Length of way: 15

```

```

a -> b
b -> c
Length of way: 17

```

```

a -> d
d -> c
Length of way: 18

```

```

a -> b
b -> d
d -> e
e -> c
Length of way: 27

a -> b
b -> d
d -> c
Length of way: 30

false.

```

Жадібний алгоритм

При пошуку шляху можна враховувати наскільки близько від фінішного стану (у сенсі якогось критерію) знаходиться поточний стан. Цей критерій називається евристичною функцією або евристикою. Він ставить у відповідність двом станам деяке число, яке характеризує «відстань» між ними. Наприклад, для відомої задачі про переміщення меблів евристикою може бути кількість предметів, які на даному етапі стоять на небажаних місцях, а для точок на площині просто геометрична відстань між ними.

У даному алгоритмі пріоритет визначається не його сумарною довжиною (вона взагалі не підраховується), а близькістю кінцевої вершини шляху і заданою фінішною вершиною.

Припустимо, є набір точок на площині, деякі з яких поєднані між собою (рис. 6).

Евристикою вважаємо геометричну відстані між точками. Задамо дану структуру як набір точок з їх координатами і набір ліній, які зв'язують їх.

```

point(a,2,5).
point(b,4,5).
point(c,0,0).
point(d,4,10).
point(e,7,8).
point(f,12,7).
point(g,14,4).

r(a,c).
r(c,b).
r(b,g).
r(g,f).
r(g,d).
r(f,d).
r(e,d).
r(a,e).

```

```
road(A,B):-r(A,B);r(B,A).
```

Шлях буде представлений списком вершин

```

prolong([Temp|Tail],[New,Temp|Tail]):-
    road(Temp,New),not(member(New,[Temp|Tail])).

```

І вводимо евристичну функцію, у якій важливими є поточна і кінцева вершини шляху.

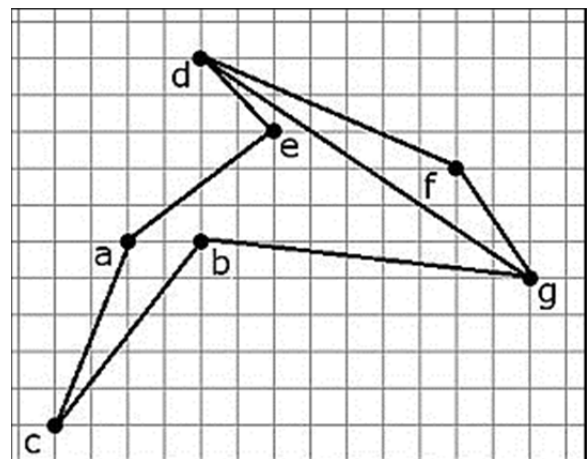


Рис. 6 – Набір точок на площині

```
wt([TempPoint|_],FinishPoint,L):-
    point(TempPoint,XA,YA),point(FinishPoint,XB,YB),
    Sum is (XA-XB)*(XA-XB)+(YA-YB)*(YA-YB), L is sqrt(Sum).
```

Предикат додавання нового шляху на потрібне місце у списку шляхів

```
%порівнюється евристики кінцевих вершин шляхів
placeone(Way,[WayH|Tail],Finish,[Way,WayH|Tail]):-
    wt(Way,Finish,A),wt(WayH,Finish,B),A<B,! .
placeone(Way,[WayH|Tail],Finish,[WayH|NewTail]):-
    placeone(Way,Tail,Finish,NewTail).
placeone(Way,[],_,[Way]).
```

В іншому пошук не відрізняється від пошуку на основі вагової функції.

Результат роботи програми:

```
?- search_grd(g,a).
```

```
g -> b
b -> c
c -> a
true .
```

Але цей результат неправильний, вірною відповіддю є

```
?- search_bst(g,a).
```

```
g -> d
d -> e
e -> a
Length of way: 21.0984
true
```

Тобто цей алгоритм не гарантує правильності результатів. Програма отримала таку відповідь, тому що за одним зі шляхів більш пріоритетним є шлях g-b, оскільки b ближча до фінішної вершини, ніж f або d. Шлях g-b можна продовжити лише одним способом g-b-c, і вершина c знов-таки стає не менш пріоритетною, ніж f або d, після чого просто завершується шлях. І ось отримане, що b близька до фінішної, c недалеко від фінішної, але те, що вони одна від одної далекі зовсім не враховувалось, що й призвело до помилки.

Але на практиці цей алгоритм можна використовувати, адже, наприклад, якщо б виконувався пошук шляху між двома реальним містами, то результат був би вірним, через рівномірне розташування міст/селищ і відповідних доріг між ними.

Завдання для практичної роботи

1. Запрограмувати методи пошуку у глибину, у ширину та жадібний пошук для незважених графів і метод пошуку найкоротшого шляху для зваженого графа.
2. Застосувати функції для пошуку шляхів для графів відповідно до варіантів.
3. Скласти звіт з роботи. Він повинен містити лістинг всіх функцій та їх застосування для пошуку у графі. Для зваженого графу вивести найкоротший шлях і всі можливі шляхи і їх довжини.

Варіанти завдань

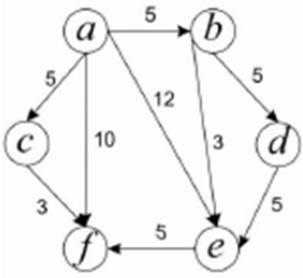
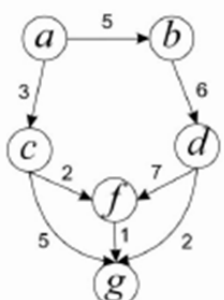
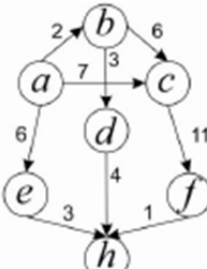
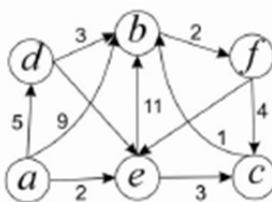
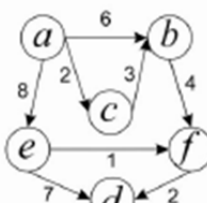
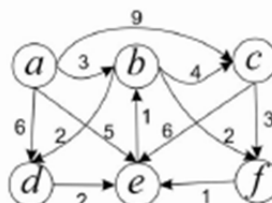
1. Незважені графи

№	Граф	№	Граф
1		2	
3		4	
5		6	
7		8	
9		10	
11		12	
13		14	

15		16	
17		18	
19		20	

2. Зважені графи

№	Граф	№	Граф
1		2	
3		4	
5		6	

7		8	
9		10	
11		12	

Контрольні питання

1. Методи пошуку у глибину і ширину.
2. Метод пошуку з ітераційним заглибленням.
3. Жадібний пошук.
4. Пошук A*.

Практична робота № 7

Віконні додатки у SWI-Prolog



Мета роботи: ознайомитися з можливостями розробки віконних додатків мовою SWI-Prolog та навчитися створювати прості віконні додатки.

Теоретичні відомості

Існує більше 20-ти програмних реалізацій мови Prolog, всі вони мають свої особливості, переваги і недоліки. Не так багато з них надають можливість створювати віконні додатки, SWI-Prolog є однією з таких реалізацій.

SWI-Prolog дозволяє створювати прості віконні додатки та багатовіконні з переходами між вікнами. Для того щоб мати можливість створювати і обробляти запити з використанням діалогових вікон, необхідно підключити бібліотеку pce:

```
:-use_module(library(pce)).
```

Розглянемо створення діалогового вікна на прикладі роботи з даними про працівників деякої організації. Спочатку створюємо основне діалогове вікно:

```
new(D,dialog('Define employee')), % D - змінна, яка відповідає об'єкту
```

```
%«вікно», dialog('Define employee') - тип об'єкта діалогове вікно, у дужках
```

```
%можна записати заголовок вікна.
```

```
...
```

```
send(D,open). %відкрити діалогове вікно, рядок є обов'язковим, %без нього вікно не з'явиться на екрані
```

Далі додаємо об'єкти, це можуть бути: кнопки, елементи для вводу тексту, списки, меню, текст або надписи та ін. Наприклад, для вводу текстової інформації елемент `text_item`:

```
new(N1, text_item(first_name)) % N1 - ім'я змінної об'єкта, % first_name - пояснючий текст
```

Для вибору з двох елементів можна використати елемент `menu`:

```
new(S, menu(sex)), % вказуємо статі
```

```
send_list(S, append, [male, female]), %додаємо значення у menu у вигляді списку
```

Для вводу числової інформації з точного діапазону можна використати елемент `int_item`:

```
new(A, int_item(age, low := 18, high := 65)) % low - найменше значення, high - найбільше
```

Якщо потрібно додати список, що розкривається, для вибору, то можна використати також елемент `menu` з параметром `cycle`:

```
new(D, menu(department, cycle)), % список, що розкривається, з назвами відділів
```

```
send_list(D, append, [research, development, marketing]) %список відділів
```

Основним елементом керування та для обробки запитів використовуються кнопки `button`. Наприклад, дві кнопки: для виходу з вікна (закрити) та обробки даних, які були введені у вікні.

```
button(cancel, message(Dialog, destroy)), %кнопка закриття діалогу
button(enter, and(message(@prolog, %кнопка, яка викликає запит
assert_employee
```

```
assert_employee,
N1?selection, %зчитування з відповідних елементів
N2?selection, %введеної інформації
S?selection,
A?selection,
D?selection),
message(Dialog, destroy))) %закриття діалогу
```

Якщо кнопку треба визначити як «за замовчуванням», це можна вказати так:
`send(Dialog, default_button, enter)`

Запит, який буде зчитувати введену інформацію про робітника і виводить повідомлення на консольний екран:

```
assert_employee(FirstName, FamilyName, Sex, Age, Depth) :-
    format('Adding ~w ~w ~w, age ~w, working at ~w~n',
    [Sex, FirstName, FamilyName, Age, Depth]).
```

Далі на рис. 7 показаний повний код для створення діалогового вікна «Define employee». На рис. 8 і 9 показаний результат запуску коду та введення даних.

```
:-use_module(library(pce)).

ask_employee :-
    new(Dialog, dialog('Define employee')),
    send_list(Dialog, append,
    [ new(N1, text_item(first_name)),
      new(N2, text_item(family_name)),
      new(S, menu(sex)),
      new(A, int_item(age, low := 18, high := 65)),
      new(D, menu(department, cycle)),
      button(cancel, message(Dialog, destroy)),
      button(enter, and(message(@prolog,
        assert_employee,
        N1?selection,
        N2?selection,
        S?selection,
        A?selection,
        D?selection),
        message(Dialog, destroy)))
    ]),
    send_list(S, append, [male, female]),
    send_list(D, append, [research, development, marketing]),
    send(Dialog, default_button, enter),
    send(Dialog, open).

assert_employee(FirstName, FamilyName, Sex, Age, Depth) :-
    format('Adding ~w ~w ~w, age ~w, working at ~w~n',
    [Sex, FirstName, FamilyName, Age, Depth]).
```

Рис. 7 – Програмний код діалогового вікна «Define employee»

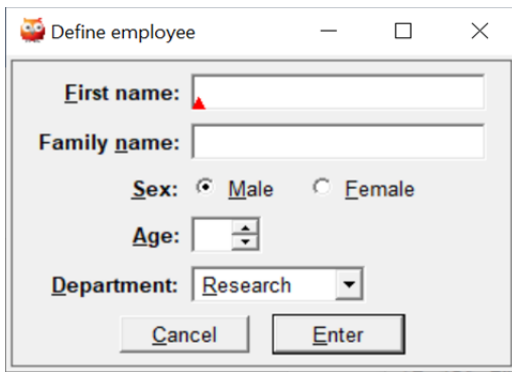


Рис. 8 – Діалогове вікно

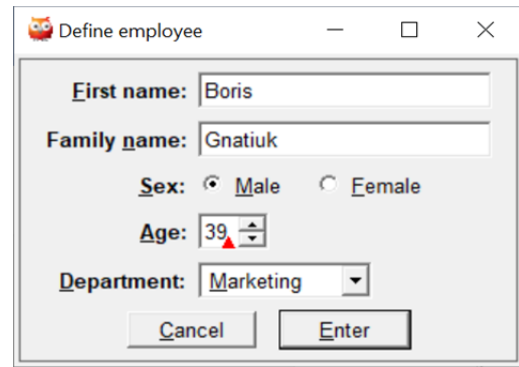


Рис. 9 – Введення даних

На рис. 10 показано повідомлення на консолі, яке з'являється після натискання кнопки «Enter».

```
?- Adding male Boris Gnatiuk, age 39, working at marketing
```

Рис. 10 – Результат роботи запиту `assert_employee`

Далі розглянемо створення віконного додатку для роботи з базою даних з інформацією про членів клубу (розглянута у практичній роботі № 5). Програмна реалізація буде мати декілька вікон: для вибору дії з базою даних, для виводу інформації на екран, для вводу нових даних тощо.

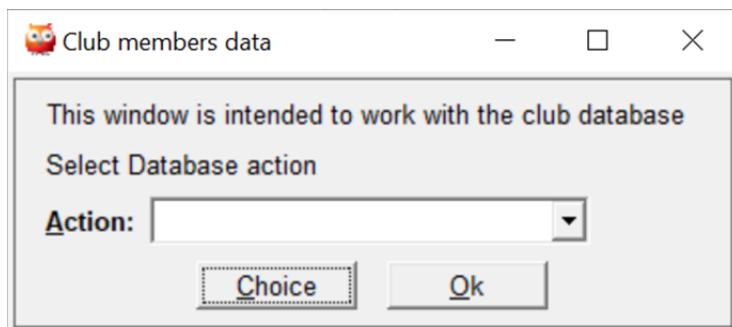


Рис. 11 – Вікно вибору дії

У першому вікні, яке з'являється при роботі з додатком, необхідно обрати потрібну дію для роботи з базою даних (рис. 11). У меню Action треба обрати одну з можливих дій: 'View the database', 'Add record to the database', 'Delete record from the database' або 'Change record in the database', і натиснути кнопку Ok.

Програмний код для цього вікна доволі зрозумілий:

```
start:-new(D,dialog('Club members data')),
    send(D,append,new(L1,label)),%пояснює повідомлення
    send(L1,append,'This window is intended to work with the
club database'),
    send(D,append,new(L2,text)),
    send(L2,append,'Select Database action'),
    send(D,append,new(Act,menu(action,cycle))),%меню вибору дій
    send_list(Act,append,[' ','View the database',
        'Add record to the database',
        'Delete record from the database',
        'Change record in the database'])),

send(D,append,button(choice,message(@prolog,work_item,Act?selectio
n))),
    send(D,append,button(ok,message(D,destroy))),
    send(D,open).
```

При натисненні кнопки «Choice» викликається запит `work_item()`, який є основним, і за обраною дією здійснює перехід до відповідних дій.

```
work_item(Item):-((Item='View the database',X is 1);
                  (Item='Add record to the database',X is 2);
                  (Item='Delete record from the database',X is 3);
                  (Item='Change record in the database',X is 4)),
    process(X).

process(X):-retractall(member/3),%завантажує базу даних
    reconsult('club_members.pl'),
    dynamic(member/3),
    item(X).%викликає запит відповідний X
```

При виборі «View the database» (переглянути базу даних), з'являється вікно з відповідною інформацією (рис. 12). Код створення цього вікна:

```
item(1):-new(D1,dialog('Information about club members')),
    send(D1,append,new(L,label)),
    send(L,append,'All information about club members'),
    send(D1,append,new(Lb,list_browser)),%поле виводу списку
    send(Lb,size,size(50,20)),%розмір поля
    send(Lb,alignment,center),
    send_list(Lb,append,['Club members\n Surname Age
Payment']),%додавання записів на поле
    member(Surname,Age,Pay_data),
    string_concat(Surname," ",S1), string_concat(Age," ",S2),
    string_concat(S1,S2,S3),string_concat(S3,Pay_data,S4),
    send_list(Lb,append,[S4]),
    send(D1,append,button(ok,message(D1,destroy))),
    send(D1,open),fail.
```

При виборі «Add record to the database» (дати запис) з'являється вікно для вводу інформації про нового члена клубу (рис. 13). Після натиснення кнопки «Add record» з'являється повідомлення, що запис був доданий (рис. 14). Код створення вікна вводу нових даних:

```
item(2):-
    new(D2,dialog('Add new club member')),

    send(D2,append,new(L,label)),
    send(L,append,'Enter data about new club member'),

    send(D2,append,new(S1,text_item(surname))),
```

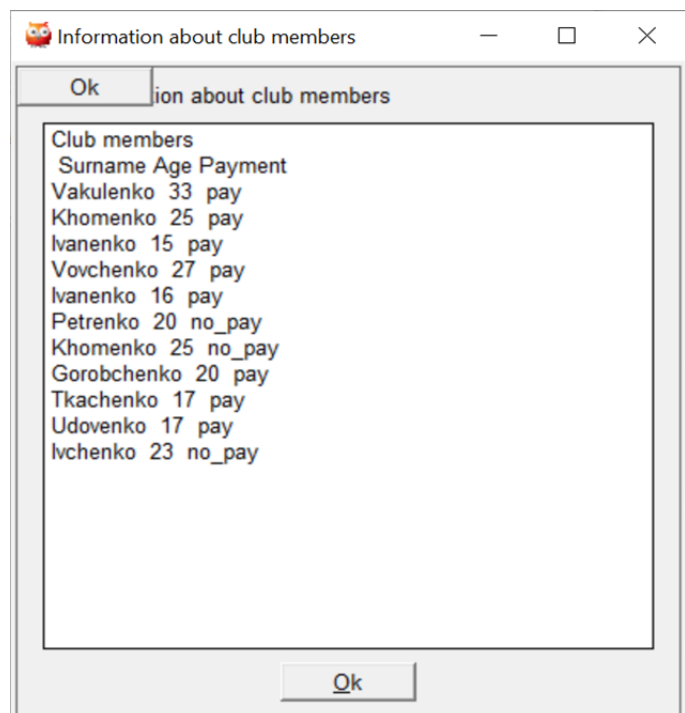


Рис. 12 – Вивід інформації з бази даних

```
send(D2,append,new(S2,int_item(age,low:=1, high:=99))),
    send(D2,append,new(S3,menu(payment))),
```

```

send_list(S3,append,['pay','no_pay']),
send(D2,append,button(add_record,message(@prolog,
                                add_record,S1?selection,
                                S2?selection,
                                S3?selection))),

send(D2,append,button(cancel,message(D2,destroy))),
send(D2,open) .

add_record(Surname,Age,Pay):-
add_member(member(Surname,Age,Pay)),
new(M,dialog('Add result')),
send(M,append,new(L,label)),
send(L,append,'New record is saved!'),
send(M,open) .

```

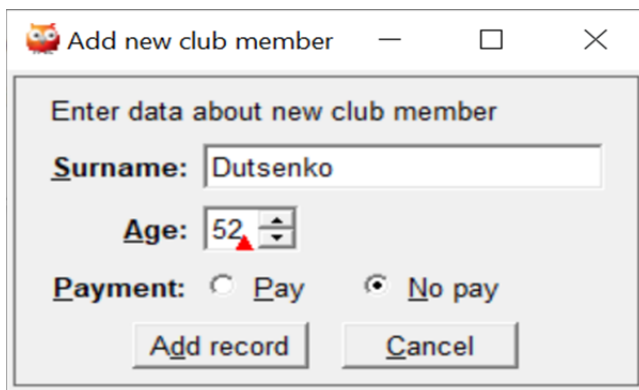


Рис. 13 – Вікно вводу нових даних

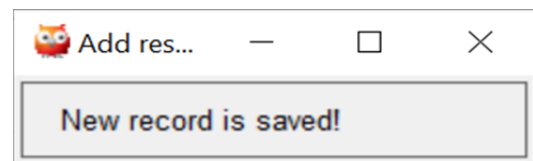


Рис. 14 – Повідомлення про збереження даних

Аналогічно створюються вікна для зміни та видалення інформації з бази даних.

Також у SWI-Prolog є можливість додавати графічні об'єкти на діалогове вікно, наприклад, зображення:

```
send(W,append, new(bitmap('web1.bmp')))
```

Крім того, можлива робота з простими графічними об'єктами (додаток Б).

Завдання до практичної роботи

1. Створити віконний додаток для роботи з базою даних за варіантом. Дій для роботи з базою даних має бути не менше трьох. Обов'язково має бути перегляд поточної інформації, додавання нової і видалення запису з бази даних.
2. Скласти звіт з роботи, в який додати лістинг програми та копії вікна з результатом виклику дій.

Варіанти

1. Конфігурація персонального комп'ютера.
2. Програмне забезпечення для персонального комп'ютера користувача.
3. Тарифні плани операторів мобільного зв'язку.

4. Обладнання для комп'ютерної мережі.
5. Програмне забезпечення для підприємства.
6. Пакети прикладних програм для розв'язання задач.
7. Програмне забезпечення для комп'ютерної мережі.
8. Наукова література у бібліотеці (для написання реферату тощо).
9. Спеціальності для навчання у вищому навчальному закладі.
10. Тури для відпочинку або подорожі.
11. Поточна успішність студентів на факультеті.
12. Результати сесії на факультеті.
13. Розклад занять у семестрі.
14. Інформація про пацієнта у лікарні.
15. Забезпеченість літературою навчального процесу.
16. Інформація про співробітників на підприємстві (декілька відділів).
17. Розклад руху поїздів на станції.
18. Художня література у бібліотеці (для школярів).
19. Інформація про навчальні заклади у регіоні.
20. За вибором студента.

Контрольні питання

1. Які об'єкти можна додавати на діалогове вікно?
2. Який предикат обов'язково має бути вказаний для відображення будь-якого вікна?
3. З якого предиката починається створення будь-якого об'єкта?
4. Як вказати, що кнопка має бути за замовчуванням?
5. Яким чином викликається запит при натисканні кнопки?
6. Чи можна викликати нове вікно при натисканні кнопки?
7. Чи є можливість роботи з графічними об'єктами?

Додатки



Додаток А

Початок роботи у SWI-Prolog. Найпростіша програма

Для встановлення SWI-Prolog на персональний комп'ютер необхідно завантажити з офіційного сайту <https://www.swi-prolog.org/download/daily/bin/> файл swipl_w32.exe (swipl_w64.exe) безкоштовно і запустити його. Після встановлення можна приступати до роботи.

Запускаємо програмне середовище SWI-Prolog. З'являється головне вікно (рис. А.1). Через головне меню можна створити, відкрити, запустити файли програм, а також у розділі Debug відстежити виконання програм.



Рис. А.1 – Головне вікно SWI-Prolog

Програма мовою SWI-Prolog складається з фактів і правил, які створюють базу знань програми, і запиту до цієї бази знань, який задає ціль пошуку рішень.

Предикати описують відношення між об'єктами, які є аргументами предиката.

Факти констатують наявність заданого предикатом відношення між вказаними об'єктами.

Наприклад, констатація факту у реченні «Еллен любить теніс» у синтаксисі SWI-Prolog має вигляд

любить('Еллен', теніс).

Де «любить» – ім'я відношення (предиката), «Еллен, теніс» – імена об'єктів (значення аргументів).

Імена предикатів (функторів) і об'єктів повинні починатися з маленької літери і можуть містити латинські літери, кирилицю, цифри і знак підкреслювання «_». Зазвичай предикатам дають такі імена, щоб вони відображали сенс відношення, наприклад: `main`, `start`, `add_file`. Два предикати можуть мати однакові імена, тоді система розпізнає їх як різні предикати, якщо вони мають різну кількість аргументів (арність). Наприклад, `like/2`, `like/3`.

Ім'я предиката може співпадати з іменем якогось вбудованого у SWI-Prolog предиката. У такому випадку при зверненні до нього буде викликаний предикат користувача, тобто визначення предиката користувачем є більш пріоритетним для інтерпретатора.

Правила описують зв'язки між предикатами.

Наприклад, речення: «Білл любить все, що любить Том», – у синтаксисі SWI-Prolog

любить('Білл',Щось):-любить('Том',Щось).

Правило В:-А відповідає імплікації $A \rightarrow B$ («Якщо А, то В»).

У загальному вигляді правило – це конструкція виду: $P_0:-P_1, P_2, \dots, P_n$, яку читаємо « P_0 істино, якщо P_1, P_2, \dots, P_n є істинними».

Предикат P_0 називається заголовком правила, вираз « P_1, P_2, \dots, P_n » – тілом правила, предикати P_i – підцілями правила. Кома між предикатами означає логічне «І». Якщо необхідно записати диз'юнктивне правило (логічне «АБО»), то правила розділяють крапкою з комою «;».

Факти і правила називаються твердженнями. Факт можна розглядати як правило, яке має заголовок і пусте тіло.

Процедура – це сукупність тверджень, заголовки яких мають однаковий функтор і однакову арність. Процедура задає визначення предиката.

У кінці речення завжди ставлять крапку, тому всі факти, правила, запити повинні закінчуватися крапкою. Необхідно зазначити також, що між іменем предиката і дужкою не повинно бути пробілів.

Змінна – це названа область пам'яті, де може зберігатися значення. Якщо змінна не пов'язана зі значенням, вона називається вільною.

Поняття змінної у логічному програмуванні відрізняється від базового поняття змінної у структурному програмуванні. Насамперед, ця відмінність полягає у тому, що змінна у SWI-Prolog, отримавши своє значення одного разу при уніфікації під час роботи програми, не може його змінити, тобто вона є, швидше, аналогом математичного поняття «змінна» – невідома величина. Змінна у SWI-Prolog не має завчасно визначеного типу даних і не може бути пов'язана зі значенням будь-якого типу даних.

Змінна у SWI-Prolog позначається як послідовність латинських літер, кирилиці й цифр, яка починається з великої літери або символу підкреслення «_». Відмітимо, що значення аргументу предиката пишеться з великої літери, його потрібно записувати в одинарних лапках.

У SWI-Prolog розрізняється реєстр, тобто імена предикатів або змінних записані великими і малими літерами будуть різні.

Розглянемо просту програму на SWI-Prolog для ілюстрації процесу створення, збереження і запуску.

Приклад 1.

```
likes('Ellen','tennis'). %Еллен любить теніс  
likes('John','football'). %Джон любить футбол  
likes('Tom','baseball'). %Том любить бейсбол  
likes('Erick','swimming'). %Ерік любить плавання  
likes('Mark','tennis'). %Марк любить теніс  
likes('Mary','dancing'). %Мері любить танці  
likes('Bill',X):-likes('Tom',X). %Білл любить те, що любить Том
```

Коментарі у рядку програми починаються з символу «%» і закінчуються кінцем рядка. Блок коментарів можна виділити спеціальними дужками /* і */.

Для того щоб набрати текст програми, використовується вбудований редактор. Щоб створити новий файл оберіть File/New у рядку меню головного вікна, у діалоговому вікні введіть ім'я нового файлу, наприклад, ex1. Для редагування вже створеного файлу з використанням вбудованого редактора використовується команда File/Edit (рис. А.2). Після того як програма була набрана або змінена, необхідно зберегти її (File/Save buffer).



Рис. А.2 – Зовнішній вигляд редактора

Червоним кольором підсвічуються предикати у заголовках, які з точки зору синтаксису SWI-Prolog є коректними. Показчик «курсор» можна використовувати для перевірки розстановки дужок тощо. Зеленим кольором виділяються коментарі, темно-червоним – змінні. Підкреслюванням виділяються предикати у тілі правила, які співпадають з предикатом заголовка, – таким чином акцентується увага на можливому зациклюванні програми.

Щоб запустити програму, спочатку її потрібно завантажити у SWI-Prolog на виконання. Це можна зробити у вікні редактора, вибравши Compile/Compile buffer. Результат компіляції відобразиться у вікні інтерпретатора SWI-Prolog. Там же будуть указані помилки, які можуть виникнути у процесі компіляції, частіше вони відображаються в окремому вікні помилок. Зазвичай перед компіляцією треба зберегти файл.

Інший спосіб завантажити вже існуючий файл – виконати команду Consult у підменю File головного вікна SWI-Prolog. У діалоговому вікні, що з'явиться,

необхідно вказати ім'я файлу і натиснути кнопку «Відкрити». Якщо у файлі, який завантажується, будуть синтаксичні помилки, він не завантажиться, і у головному вікні з'являться повідомлення про помилки.

За замовчуванням файли SWI-Prolog мають розширення .pl.

Файли також можна завантажити, використовуючи вбудований предикат

`consult` (ім'я файлу або декількох файлів).

Наприклад:

`consult(test).`

`consult([test1, test2]).` % завантаження двох файлів

`consult('test.pl').`

Для виконання завантаження цей предикат необхідно написати у головному вікні після запрошення інтерпретатора (?-), яке означає, що інтерпретатор чекає на запит.

Запит – це конструкція виду: ?- P_1, P_2, \dots, P_n , яку читаємо «Чи є вірним P_1 і P_2 і ... P_n ?». Предикати P_i є підцілями запиту.

Запит є способом запуску механізму логічного виводу, тобто фактично запускає програму.

Для перевірки речень завантаженої бази знань можна використати вбудований предикат `listing`.

Приклад 2.

Введемо кілька запитів до створеної бази знань.

?-likes('Bill', 'tennis').

?-likes(Who, 'tennis').

?-likes('Mark', What), likes('Ellen', What).

?-likes(Who, What).

?-likes(Who, _).

Результати запитів показані на рис. А.3 (а – в).

Якщо необхідно продовжити пошук у базі за цим же запитом і отримати альтернативні розв'язки, то вводиться крапка з комою «;».

Якщо необхідно перервати виконання запиту, необхідно ввести `b`.

Якщо необхідно повторити один з попередніх запитів, можна скористатись клавішами «Стрілка вгору» або «Стрілка вниз».

Перезавантажити файли, які були змінені у зовнішньому редакторі, можна використовуючи вбудований предикат `make`. Наприклад, так: ?-make.

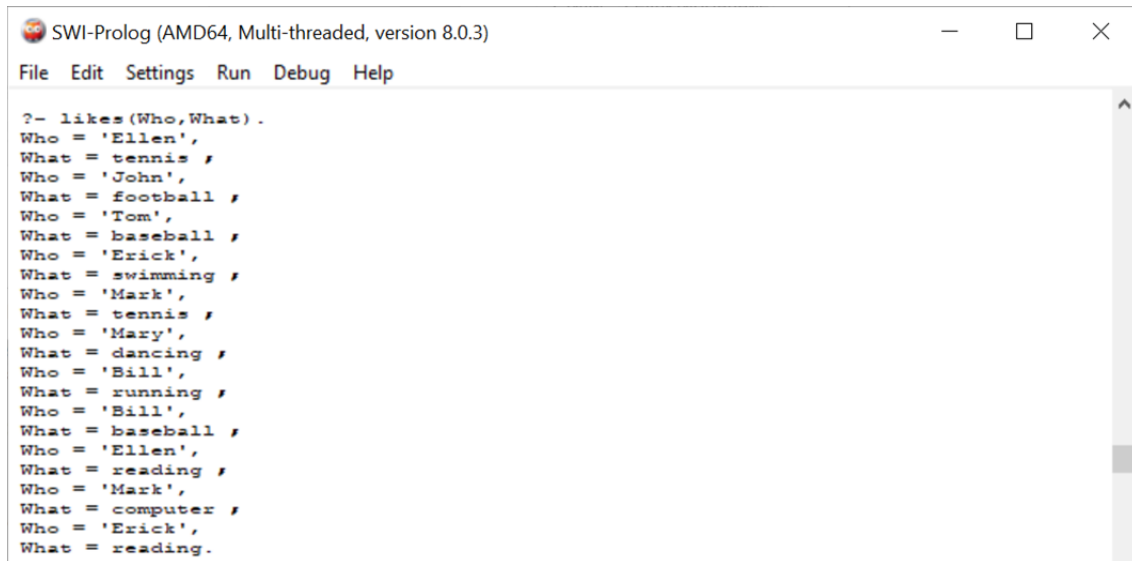


```
SWI-Prolog (AMD64, Multi-threaded, version 8.0.3)
File Edit Settings Run Debug Help
?- likes('Bill','tennis').
false.

?- likes(Who,'tennis').
Who = 'Ellen' ;
Who = 'Mark' ;
false.

?- likes('Mark',What),likes('Ellen',What).
What = tennis ;
false.
```

a)



```
SWI-Prolog (AMD64, Multi-threaded, version 8.0.3)
File Edit Settings Run Debug Help

?- likes(Who,What).
Who = 'Ellen',
What = tennis ;
Who = 'John',
What = football ;
Who = 'Tom',
What = baseball ;
Who = 'Erick',
What = swimming ;
Who = 'Mark',
What = tennis ;
Who = 'Mary',
What = dancing ;
Who = 'Bill',
What = running ;
Who = 'Bill',
What = baseball ;
Who = 'Ellen',
What = reading ;
Who = 'Mark',
What = computer ;
Who = 'Erick',
What = reading.
```

б)



```
SWI-Prolog (AMD64, Multi-threaded, version 8.0.3)
File Edit Settings Run Debug Help

?- likes(Who,_).
Who = 'Ellen' ;
Who = 'John' ;
Who = 'Tom' ;
Who = 'Erick' ;
Who = 'Mark' ;
Who = 'Mary' ;
Who = 'Bill' ;
Who = 'Bill' ;
Who = 'Ellen' ;
Who = 'Mark' ;
Who = 'Erick'.
```

в)

Рис. А.3 – Результат запитів

Додаток Б**Приклади Prolog-програм****Приклад 1.** Організація логічного виводу

Бутсі – коричнева кішка. Корні – чорна кішка. Мак – рижа кішка. Флеш, Ровер, Спот – собаки, Ровер – рижа, а Спот – біла. Усі тварини, якими володіють Том і Кейт, мають родословні. Том володіє усіма чорними і коричневими тваринами, а Кейт володіє усіма собаками небілого кольору, які не є власністю Тома. Алан володіє Мак, якщо Кейт не володіє Бутсі, і якщо Спот не має родословної. Флеш – плямистий собака. Визначити, які тварини не мають господарів.

Розв'язок задачі мовою SWI-Prolog

```
cat('Butsi'). %предикат cat – кішка, аргумент – ім'я кішки
cat('Korni').
cat('Mac').
dog('Rover'). %предикат dog – собака, аргумент – ім'я собаки
dog('Flesh').
dog('Spot').
color('Butsi','brown').%предикат, який визначає колір тварини
color('Korni','black').
color('Mac','red').
color('Rover','red').
color('Spot','white').
color('Flesh','black_white').
animal(X):-cat(X);dog(X).%предикат який задає правило,
визначення тварини
rodoslovnaya(X):-animal(X),have('Tom',X).%правило визначення
тварини з родословною
rodoslovnaya(X):-animal(X),have('Kate',X).
have('Tom',X):-color(X,'black');color(X,'brown').%правило, яке
визначає тварин Тома
have('Kate',X):-
dog(X),not(color(X,'white')),not(have('Tom',X)).%ким володіє Кейт
have('Alan','Mac'):-
not(have('Kate','Butsi')),not(rodoslovnaya('Spot')).% чи володіє
Алан Маком
no_owner:-animal(X),not(have(_,X)),write(X).%правило, яке визначає
тварину без господаря
```

Виклик запиту

```
?- no_owner.
Spot
true.
```

Відповідь: господаря не має Спот.

Приклад 2. Задача Ейнштейна

П'ять різних людей живуть у п'яти різних будинках, палять п'ять різних марок цигарок, люблять п'ять різних видів тварин, п'ють п'ять різних напоїв. Питання: хто любить рибок?

Підказки:

- Норвежець живе у першому будинку.
- Англієць живе у червоному будинку.
- Зелений будинок знаходиться зліва від білого.
- Данець п'є зелений чай.
- Той, хто палить Marlboro, живе поряд з тим, хто любить кішок.
- Той, хто живе у жовтому будинку, палить Dunhill.
- Німець палить Rothmans.
- Той, хто живе у центрі, п'є молоко.
- Сусід, того хто палить Marlboro, п'є воду.
- Той, хто палить Pall Mall, любить пташок.
- Швед любить собак.
- Норвежець живе поряд із синім будинком.
- Той, хто любить коней, живе у синьому будинку.
- Той, хто палить Winfield, п'є вино.
- У зеленому будинку п'ють каву.

Розв'язок задачі мовою SWI-Prolog із використанням списків:

```
einstein :-
    /* 0. Усього 5 будинків */
    Houses = [_,_,_,_,_],
    /* 1. Норвежець живе у першому будинку. */
    nth1(1, Houses, [norwegian,_,_,_,_]),
    /* 2. Англієць живе у червоному будинку. */
    member([englishman,_,_,_red], Houses),
    /* 3. Зелений будинок знаходиться зліва від білого. */
    nextto([,_,_,_green], [_,_,_,_white], Houses),
    /* 4. Данець п'є зелений чай. */
    member([dane,_,_tea,_], Houses),
    /* 5. Той, хто палить Marlboro, живе поряд з тим, хто
любить кішок. */
    neighbors([,_,_marlboro,_,_], [_,_cat,_,_,_], Houses),
    /* 6. Той, хто живе у жовтому будинку, палить Dunhill. */
    member([,_,_dunhill,_yellow], Houses),
    /* 7. Німець палить Rothmans. */
    member([german,_rothmans,_,_], Houses),
    /* 8. Той, хто живе у центрі, п'є молоко. */
    nth1(3, Houses, [_,_,_milk,_]),
    /* 9. Сусід, того хто палить Marlboro, п'є воду. */
    neighbors([,_,_marlboro,_,_], [_,_,_water,_], Houses),
    /* 10. Той, хто палить Pall Mall, любить пташок. */
    member([,_bird,pallmall,_,_], Houses),
    /* 11. Швед любить собак */
    member([swede,dog,_,_,_], Houses),
    /* 12. Норвежець живе поряд із синім будинком. */
    neighbors([norwegian,_,_,_,_], [_,_,_,_blue], Houses),
    /* 13. Той, хто любить коней, живе у синьому будинку. */
    member([,_horse,_,_blue], Houses),
    /* 14. Той, хто палить Winfield, п'є вино. */
    member([,_,_winfield,wine,_], Houses),
```

```

/* 15. У зеленому будинку п'ють каву. */
member([_,_,_,coffee,green], Houses),
/* Увага, питання: у кого рибки?*/
member([Owner,fish,_,_,_], Houses),
/* Отримаємо розв'язок */
print('Owner of the fish: '), print(Owner), nl,
print('Full Solution: '), print(Houses), nl.
/* Допоміжне правило: */
neighbors(X, Y, List) :- nextto(X, Y, List) ; nextto(Y, X, List) .

```

Результат роботи програми

```

'Owner of the fish: 'german
'Full      Solution:'      [[norwegian,cat,dunhill,water,yellow],
[dane,horse,marlboro,tea,blue],
[englishman,bird,pallmall,milk,red],
[german,fish,rothmans,coffee,green],
[swede,dog,winfield,beer,white]]

```

Відповідь: рибок любить німець.

Приклад 3. Логічний вивід із використанням списків

Студенти факультету комп'ютерних наук Артур, Василь, Надія і Еліна на канікулах вирушили до м. Дрезден на екскурсію. У дорозі стало відомо, що їм подобаються твори різних митців: Моне, Рафаеля, Родена і Рембрандта. Тому, приїхавши, троє з них вирушили до різних музеїв (Галерея нових майстрів, Галерея старих майстрів, Зібрання скульптур), а Василь вирішив піти до фізико-математичного салону, оскільки тих картин, які йому подобаються, у музеях Дрездена немає. Відомо, що: у Зібранні скульптур можна знайти роботи Родена, у Галереї старих майстрів немає картин імпресіоністів. Надія цікавиться імпресіоністами і більш за всіх їй подобається Моне, а Еліна надає перевагу Рафаелю. Визначте, куди пішов кожен зі студентів та які їхні смаки.

Розв'язок задачі мовою SWI-Prolog

```

student('Vasil').
student('Artur').
student('Nadia').
student('Elina').
painter('Mone').
painter('Rafael').
painter('Roden').
painter('Rembrandt').
museum('Galerie Neue Meister').
museum('Gemäldegalerie Alte Meister').
museum('Skulpturensammlung').
museum('Mathematisch-Physikalischer Salon').

unique([]):-!. %перевірка, що у списку немає елементів, які
повторюються
unique([Head|Tail]):-
    member(Head, Tail), !, fail;
    unique(Tail).

choice(Students):-
    Students=[like(Name1, Museum1, Painter1),
              like(Name2, Museum2, Painter2),

```



```

high:=300,
default:=150))]],
send(DW, append, new(X, int_item(x_coord, default:=10))),
send(DW, append, new(Y, int_item(y_coord, default:=10))),
send_list(DW, append, [button('намалювати прямокутник',
message(@prolog, mybox, Picture,
Width?selection,
Height?selection,
X?selection,
Y?selection)),
button('намалювати еліпс',
message(@prolog, myellipse, Picture,
Width?selection,
Height?selection,
X?selection,
Y?selection)),
button('витерти', message(Picture, clear))]),
send(DW, append, button(exit, and(message(DW, destroy),
message(Picture, destroy)))),
send(Picture, open),
send(DW, open).

mybox(Picture, Width, Height, X, Y):-send(Picture, display,
new(Box, box(Width, Height)),
point(X, Y)),

send(Box, colour, colour(green)),
send(Box, fill_pattern, colour(red)).

myellipse(Picture, Width, Height, X, Y):-send(Picture, clear),
send(Picture, display, new(Ellip, ellipse(Width, Height)),
point(X, Y)),
send(Ellip, fill_pattern, colour(yellow)).

```

Приклад запуску вікна

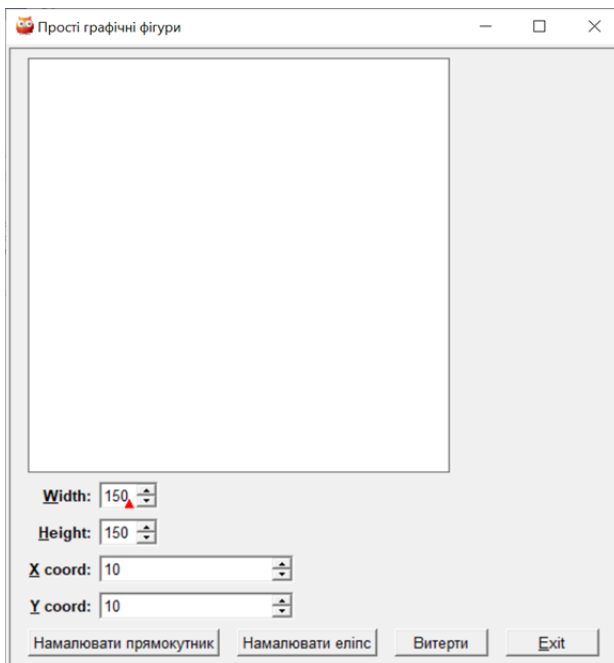


Рис. Б.1 – Вікно після запуску програми

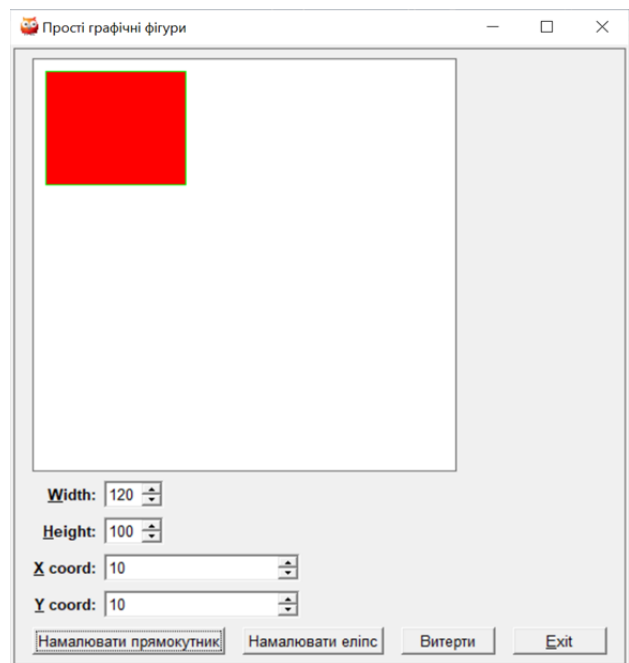


Рис. Б.2 – Зображення прямокутника

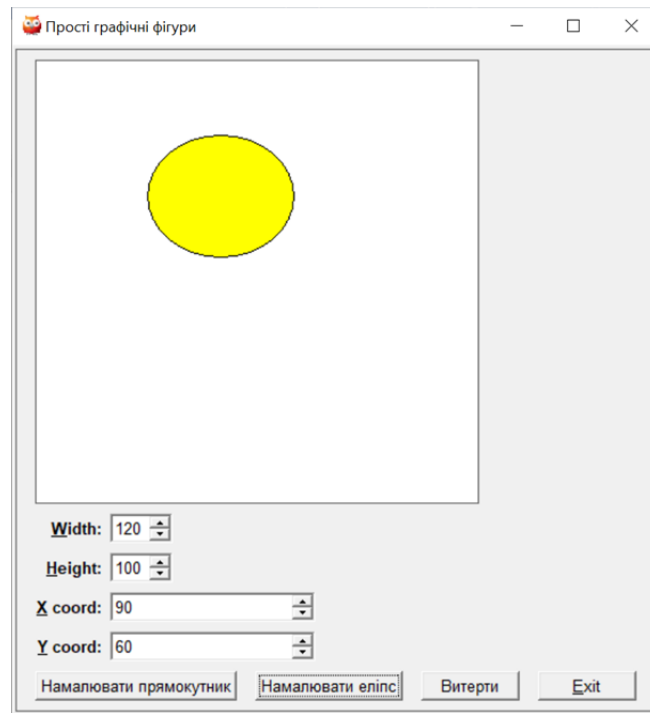


Рис. Б.3 – Зображення еліпса

Додаток В**Вбудовані предикати і оператори SWI-Prolog**

Предикат/Оператор	Опис
write(Term)	вивести Term у потік виводу
read(Term)	вилучити з потоку виводу терм і уніфікувати його з Term. Введення закінчується крапкою
tell(File)	відкрити файл для запису і перевести в нього потік виводу
told	перевести потік виводу у стандартний вивід (закрити файл)
see(File)	відкрити файл для читання і перевести в нього потік введення
seen	перевести потік введення у стандартний (не з файлу) режим
append(List1, List2, List3)	успішний, коли список List3 уніфікуємо з об'єднанням списків List1 і List2. Усі аргументи можуть бути вільними змінними. Результат – уніфікація відповідних списків
append(ListOfList, List)	успішний, коли об'єднання списку зі списків ListOfList уніфікуємо зі списком List. Результат – уніфікація відповідних списків
member(Elem, List)	успішний, коли Elem уніфікуємо з одним з елементів списку List. Результат – уніфікація відповідного елементу списку з Elem
nextto(X, Y, List)	успішний, коли Y йде за X у списку List
delete(List1, Elem, List2)	видалити усі елементи списку List1, які уніфікуються з Elem. Результат розміщується у List2.
select(Elem, List, Rest)	успішний, коли Rest є списком List з видаленим елементом Elem. Тобто цим предикатом можна видаляти і вставляти елементи списку. Наприклад: ?-select[a, L, [1, 2, 3]]. L=[a, 1, 2, 3]; L=[1, a, 2, 3]; L=[1, 2, a, 3]; L=[1, 2, 3, a]; false.
nth0(Index, List, Elem)	успішний, коли елемент списку List з номером Index уніфікуємо з Elem. Відлік номерів елементів починається з 0
nth1(Index, List, Elem)	як і nth0/3, але відлік номерів починається з 1
last(List, Elem)	успішний, коли Elem уніфікується з останнім елементом списку List. Якщо хвіст списку List не визначений, то під час бектрекінгу хвіст буде збільшуватися

reverse(List1, List2)	змінює порядок елементів List1 і уніфікує результат з List2
permutation(List1, List2)	успішний, коли список List1 створений зі списку List2 перестановкою елементів
sumlist(List, Sum)	уніфікує Sum з сумою елементів списку List
max_list(List, Max)	уніфікує Max з максимальним елементом списку List
min_list(List, Min)	уніфікує Min з мінімальним елементом списку List
IntExpr1 mod IntExpr2	остача від ділення IntExpr1 на IntExpr2
IntExpr1 // IntExpr2	ціла частина від ділення IntExpr1 на IntExpr2
sleep(Time)	призупинення виконання на Time секунд
Term=..List	вираз істинний, коли терм Term ініціюється з термом відповідного списку List, головою якого є функтор терма, а хвосту є списком аргументів терму. Наприклад: $?-a(b(1), 2, 3, X) = \dots L.$ $L = [a, b(1), 2, 3, X].$ $?-a(b(1), 2, 3, X) = \dots [a, b(X) _].$ $X = 1.$ $?-a(Y, 2, 3, X) = \dots [a, b(2) _].$ $Y = b(2).$ $?-Term = \dots [a, b, c(4), 1, 2, 3, X].$ $Term = a(b, c(4), 1, 2, 3, X).$ Таким чином за допомогою цього оператора можна конструювати терми з елементів і «розбивати» їх на елементи.
call()	«запустити на виконання» предикат, який зберігається у Goal. Цей предикат може бути потрібен, наприклад, після конструювання предиката попереднім оператором. Наприклад: $?-assert(a(1)).$ $?-read(Y), X = \dots [a, Y], call(X).$ $: 1.$ $Y = 1,$ $X = a(1).$ $?-read(Y), X = \dots [a, Y], call(X).$ $: 23.$ $false.$
op(Precedence, Type, Name)	об'ява операторної форми з іменем Name, пріоритетом Precedence і типом Type для предиката з іменем Name. Нові оператори можна вводити для збільшення зручності запису потрібних предикатів
trace(Pred)	установити точку трасування предиката з іменем Pred, тобто під час трасування буде відображатися кожна подія, пов'язана з предикатом Pred

trace(Pred, Ports)	<p>установити точку трасування предиката з іменем Pred, яка активується за подіями Ports. Події можуть бути таких типів:</p> <p>fail – подія, яка відповідає неуспішному виклику предиката з іменем Pred;</p> <p>call – подія, яка відповідає першому виклику предиката з іменем Pred;</p> <p>redo – подія, яка відповідає повторному виклику предиката з іменем Pred;</p> <p>exit – подія, яка відповідає успішному закінченню виклику предиката з іменем Pred.</p> <p>Параметр Ports може набувати значення типів повідомлень з префіксами, які відповідають додаванню і видаленню події (порту) з точки трасування, а також списки цих значень.</p> <p>Наприклад, запит</p> <pre>?-trace(foo/2,+),trace(foo/2,[+call,-call]).</pre> <p>додає у точку трасування пов'язану з предикатом foo/2 подію fail. Для додавання події використовують префікс +, для видалення -. Крім того, можна оперувати відразу зі всіма типами подій, використовуючи позначення all. Наприклад: trace(foo/2,+all).</p>
tracing	успішний, якщо викликається з режиму трасування
notrace	вихід з режиму трасування
guitracer, gtrace	увімкнути графічний режим трасування. Вікно графічного режиму трасування вмикається при зустрічі контрольної точки (spy-point)
noguitracer	вимкнути графічний режим трасування
debug	увімкнути режим налагодження. При запитах до програми у режимі налагодження виводяться усі події, які відбуваються під час виконання запиту, пов'язані з встановленими точками налагодження
nodebug	вимкнути режим налагодження
debugging	показати точки трасування, які відстежуються, і контрольні точки
spy(Pred)	установити контрольну точку, пов'язану з предикатом Pred
nospy(Pred)	прибрати контрольну точку, пов'язану з предикатом Pred
nospyall	прибрати усі контрольні точки

Список літератури



1. Luger G. F. Artificial Intelligence. Structures and Strategies for Complex Problem Solving / George F. Luger. – Pearson Education Limited, 2005. – 903 p.
2. Конверський А. Є. Сучасна логіка (класична та некласична) / А. Є. Конверський. – К. : Центр учбової літератури, 2017. – 294 с.
3. Матвієнко М. П. Математична логіка та теорія алгоритмів : навч. посіб. / М. П. Матвієнко, С. П. Шаповалов. – Київ : Ліра-К, 2015. – 211 с.
4. Шкільняк С. С. Математична логіка. Основи теорії алгоритмів : навч. посіб. / С. С. Шкільняк. – К. : ДП «Вид. дім "Персонал"», 2009. – 280 с.
5. Russel S. Artificial Intelligence. A modern approach / Stuart J. Russel, Peter Norvig. – Pearson Education Limited, 2003. – 1170 p.
6. Bratko I. Prolog Programming for Artificial Intelligence / Ivan Bratko. – Pearson Education, 2012. – 560 с.
7. Jones M. T. Artificial Intelligence. Application Programming / M. Tim Jones. – Dreamtech Press, 2004. – 473 p.
8. Глинський Я. М. Штучний інтелект. Інтелектуальні роботи / Я. М. Глинський, В. А. Рязька. – Львів : Деол, 2002. – 168 с.
9. Jackson P. Introduction to Expert Systems / Peter Jackson. – Addison Wesley, 1990. – 352 p.
10. Спірін О. М. Початки штучного інтелекту : навчальний посібник / О. М. Спірін. – Житомир : Вид-во ЖДУ, 2004. – 172 с.
11. Programming in XPC/Prolog / J. Wielemaker, A. Anjewierden. – University of Amsterdam, 2005. – 289 p.
12. Зайченко Ю. П. Основи проектування інтелектуальних систем : навчальний посібник / Ю. П. Зайченко. – Київ : Слово, 2004. – 352 с.
13. Bramer M. Logic programming with Prolog / Max Bramer. – Springer, 2005. – 224 p.
14. Ulle Endriss. An introduction to Prolog Programming. Lecture Notes / Endriss Ulle. – University of Amsterdam, 2014. – 66 p.
15. Clocksin William F. Programming in Prolog / William F. Clocksin, Christopher S. Mellish. – Berlin : Springer-Verlag, 1994.
16. Covington Michael A. Prolog Programming in Depth / Michael A. Covington. – Prentice Hall, 1996. – 516 p.
17. Merritt Dennis. Expert Systems in Prolog / Dennis Merritt. – Springer, 2011. – 358 p.
18. Nilsson Ulf. Logic, Programming and Prolog / Ulf Nilsson, Jan Maluszynski. – John Wiley & Sons Inc., 2000. – 296 p.