МЕЛІТОПОЛЬСЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ
УНІВЕРСИТЕТ
ІМЕНІ БОГДАНА ХМЕЛЬНИЦЬКОГО

# МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З КУРСУ «ОСНОВИ ЛОГІЧНОГО ТА ФУНКЦІОНАЛЬНОГО ПРОГРАМУВАННЯ»

Мелітополь – 2018

**ББК** 32.973-01я7
**УДК** 004.89(072)

Методичні рекомендації до виконання лабораторних робіт з курсу «Основи логічного та функціонального програмування» / укл.: С.В.Шаров – Мелітополь: РВЦ МДПУ, 2018. – 52 с.

**Рецензенти:**
**В.М. Малкіна** – д.т.н., професор, завідувач кафедри прикладної геометрії і інформаційних технологій проектування ім. академіка В.М. Найдиша Таврійського державного агротехнологічного університету;
**Ю.О. Сіціліцин** – старший викладач кафедри інформатики і комп'ютерного моніторингу Мелітопольського інституту екології і соціальних технологій.

Дані методичні рекомендації призначені для студентів, які навчаються за спеціальностями 122 Компютерні науки та 14.09 Середня освіта (Інформатика). Рекомендації містять теоретичні відомості та завдання до лабораторних робіт згідно робочої програми курсу «Основи логічного та функціонального програмування».

Рекомендовано до друку рішенням науково-методичної ради Мелітопольського державного педагогічного університету імені Богдана Хмельницького.
Протокол №2 від 13.12.17

**ЗМІСТ**

# ВСТУП

Сьогодні перспективними напрямками розвитку комп'ютерної техніки є використання систем із штучним інтелектом. Такі системи отримали назву інтелектуальні системи. Основою для розробки інтелектуальних систем стали мови логічного та функціонального програмування. Яскравим представником таких інструментальних засобів є мова Turbo Prolog та Elixir.

Основною метою викладання дисципліни "Основи логічного та функціонального програмування" є формування у студентів базових знань та практичних навичок з мови програмування Turbo Prolog та Elixir, ознайомлення студентів із теоретичними відомостями про логічне програмування та способи його використання у майбутній професійній діяльності.

Завдання вивчення курсу "Основи логічного та функціонального програмування" полягає у теоретичній та практичній підготовці майбутніх фахівців з таких питань: надання навичок програмування на мові Turbo Prolog та Elixir; ознайомлення студентів з різноманітними підходами до створення експертних систем; виявлення зв'язку інформатики з іншими науками; розвиток інформаційної культури студентів.

Структура методичних рекомендацій складається зі вступу, завдань до лабораторних робіт, основної та додаткової літератури. Кожна лабораторна робота містить тему та мету роботи, пояснення до виконання, послідовну інструкцію для виконання практичних завдань, коротку відповідь на контрольні питання, інструкцію до виконання практичних завдань. У теоретичному матеріалі подані приклади виконання практичних завдань, що дозволить студенту краще засвоїти дисципліну.

Методичні рекомендації допоможуть студентам у виконанні практичних завдань з курсу "Основи логічного та функціонального програмування" Під час виконання лабораторних робіт студенти ознайомляться з основами логічного та функціонального програмування, навчаться використовувати предикати арифметичних операцій, обробляти списки, оголошувати та використовувати рекурсію, зчитувати інформацію з файлу та записувати оброблені дані у файл, працювати з предикатами для обробки баз даних, обробляти графи.

## ЛАБОРАТОРНА РОБОТА №1

**Тема**: Вступ у мову логічного програмування Turbo Prolog

**Мета**: засвоїти знання про визначення та призначення мови логічного програмування Turbo Prolog, опанувати основними принципами логічного програмування.

**Пояснення** до лабораторної роботи: студенту необхідно знати теоретичний матеріал першої лекції та підготовитися до відповіді за запропонованими викладачем питаннями. Показати складені програми викладачу.

*Література*: [1], [2], [3], [4], [5].

### Питання для перевірки

1) Структура мови Турбо-Пролог?
2) Форма запису фактів у Турбо-Пролог?
3) Запис імен, типів даних у Турбо-Пролог?
4) Запис констант і змінних у Турбо-Пролог?
5) Перелічите основні програмні секції Турбо-Прологу?
6) Призначення та структура секції Domains?
7) Призначення та структура секції Predicates?
8) Призначення та структура секції Database?
9) Призначення та структура секції Clauses?
10) Призначення та структура секції Goal?
11) Як створити новий файл у Турбо-Пролог?
12) Як зберегти на диск створений у Турбо-Пролог файл?

### Короткі теоретичні відомості

Turbo Prolog – це декларативна мова, програми на якій містять оголошення логічних взаємозв'язків, необхідних для вирішення завдання. У Turbo Prolog розглядаються відносини між твердженнями і об'єктами, характерні для логіки предикатів.

Програма на Turbo Prolog включає певні розділи, не всі з яких є обов'язковими:

Domains /*(домени) – розділ оголошень*/;

Database /* описи предикатів динамічної бази даних */

Predicates /* описи предикатів */

Clauses /* твердження – факти і правила */

Goal /* цільове твердження */

У програмі обов’язково повинні бути розділи predicates і clauses. Розділ domains нагадує оголошення даних в традиційних (імперативних) мовах, наприклад таких, як Паскаль і Сі.

У Prolog підтримується шість типів даних:

| | |
|---|---|
| Symbol | Послідовність букв, цифр і знаків підкреслення, що починається з малої літери або укладена в лапки |
| String | Будь-яка послідовність символів, що укладена в лапки |
| Char | Окремий символ, укладений в апострофи |
| Integer | Ціле число в діапазоні від -32768 до 32767 |
| Real | Будь-яке число (може бути представлене в експонентному форматі) |
| File | Ім’я файлу |

*Константи* Turbo Prolog можуть бути записані різними способами:

а) або з маленької букви (крім кирилиці):

fact1, summa, person;

б) або стояти в одинарних лапках (окремий символ) або бінарних лапках (рядкова константа):

'c' , "summa=", "сума";

в) або вони можуть числами (цілими або речовинними):

25, 0.5, 3.2 e-4 .

Таким чином, константи можуть бути кожного зі стандартних типів Turbo Prolog. У програмі тип констант явно не вказується.

*Змінні* – це ланцюжок, що складаються з букв, цифр і символу підкреслення. Вони починаються із прописної букви або із символу підкреслення:

X, Summa, List_of_members, _x23.

Змінна може мати один зі стандартних типів, або її тип визначається в секції *domains*. Можна також використовувати так звану анонімну змінну, яка записується у вигляді одного символу підкреслення.

Програма на Пролозі складається з *пропозицій* (або тверджень). Кожна пропозиція закінчується крапкою. Пропозиції бувають двох видів: *факти та правила.*

Число аргументів предиката називається його *арностью.* Ім’я предиката складається з букв, цифр, знаків підкреслення. Воно повинно починатися з рядкової букви або знаку підкреслення.

*Факт* констатує, що між об’єктами виконано деяке відношення. Він складається тільки із заголовка. Факт – дійсне твердження.

*Правило* – це пропозиція, істинність якої залежить від істинності предикатів, що становлять тіло правила. Формат:

<правило> ::= <предикат> :- <предикат> | [, <предикат>] * .

*Уніфікація* (ототожнення) – механізм зіставлення цільового предиката з базою даних і конкретизації (зіставлення) змінних.

Механізм уніфікації Turbo Prolog використовує наступні правила при узгодженні цілей:

1. Вільні змінні можуть бути уніфіковані з будь-яким термом.
2. Атоми та константи можуть бути уніфіковані тільки самі з собою.
3. Структурний терм може бути уніфікований з іншим структурним термом, якщо обидва вони мають однакові функтори та рівне число компонент, при цьому константи, використовувані у якості компонент можуть бути зв'язані тільки самі з собою або з вільною змінною.

Приклад: Два терми data(D,M,1993) і data(D1,"май",Y) можуть бути співставлені. Результатом їх уніфікації (зіставлення) є ототожнення змінних: D = D1  M = "травень"  Y = 1993

## Приклади виконаних завдань

*Приклад 1.* Скласти програму, яка зберігає та обробляє інформацію про дочку та її матір.

```
мама("Наташа","Даша").
мама("Даша","Маша").
бабуся(X,Y):- мама(X,Z), мама(Z,Y).
```

Вивести ім’я мами Даши: мама(X,Даша).

Питання про ім'я дочки Наташі записується у вигляді: мама(Наташа,X).

Можна попросити систему Turbo Prolog знайти імена всіх відомих їй мам і дочок, поставивши питання: мама(X,Y).

Якщо треба одержати тільки імена всіх мам, можна скористатися анонімною змінною і записати питання: мама(X,_).

Нарешті, якщо треба одержати відповідь на питання: чи є інформація про людей, що знаходяться у відношенні "мама - дочка", то його можна сформулювати у вигляді: мама(_,_).

*Приклад 2.* Скласти програму “може купити”. Додати предикат

can_buy(X, Y) та пов’язати його з іншими предикатами.

```
predicates
  can_buy(symbol, symbol) /* відношення "може купити" */
  person(symbol)      /* відношення "суб'єкт */
  car(symbol)       /* відношення "марка автомобіля" */
  likes(symbol, symbol)
  for_sale(symbol)    /* відношення "продається" */
clauses
  can_buy(X, Y):-         person(X), car(Y), likes(X, Y), for_sale(Y).
  person(kelly). person(judy).
  car(lemon). car(hot_rod).
  likes(kelly, hot_rod). likes(judy, pizza).
  for_sale(pizza). for_sale(lemon). for_sale(hot_rod).
```

*Приклад 3.* Задано наступні предикати: dog, cat, parent

```
domains
        person=symbol
predicates
        dog(person)
        cat(person)
        parent(person,person)
clauses
        dog(linda).
        dog(gera).
        dog(X):-parent(X,Y),dog(Y).
        cat(vacya).
        cat(black).
        cat(blue).
        parent(rex,linda).
        parent(boxer,linda).
        parent(vacya,black).
        parent(masha,blue).
```

Виконати й пояснити наступні запити:

1. boxer є собакою? dog(boxer).
2. gray є котом? cat(gray).
4. чи є gray батьком (parent) для masha? parent(gray, masha).
5. запросити імена всіх кішок? cat(X).

## Завдання до виконання

*Завдання 1.* Скласти програму, що містить наступні факти

```
male(vit).
male(vik).
male(ivan).
female(valya).
female(klara).
mother(klara, vit).
mother(klara, vik).
father(ivan, valya).
father(ivan, vik).
```

1. Записати правило для визначення відношення батько(parent), сестра (sister), брат(brother). Виконати запити до програми.

2.Окрім споріднених відносин parent (батько) і ancestor (предок) програма повинна містити хоч би одне з наступних відносин: brother (брат); sister (сестра); grand-father (дідусь); grand-mother (бабуся); uncle (дядько);

3. Закоментувати внутрішню мету. Запустити програму на виконання. Задати зовнішню мету: знайти всіх братів і сестер.

4. Вставити на початку програми директиву трасування trace. Запустіть програму. Покрокове виконання програми відбувається при натисненні клавіші F10. Простежте за виконанням програми. Перервати виконання програми можна у будь-який момент натисненням клавіші ESC.

4. Сформулювати у вигляді зовнішніх цілей питання:

Хто є батьком?

Чи є у Сема дитина?

Хто діти Марії?

У кого є сестра?

*Завдання 2.* Сформувати факти, що зазначають професію й місто проживання клієнта. Запросити із клавіатури місто й професію, і видати всіх клієнтів, що задовольняють запиту.

*Завдання 3.* Відомі наступні відомості про автомобілі: марка машина, рік випуску, колір, ціна.

Скласти програму для зберігання та обробки зазначених характеристик відповідно до правил оформлення програми на Turbo Prolog.

Необхідно довідатися (поставити запитання програмі):

а) Чи є автомобіль червоного кольору, ціна якого менше 1000$?

б) Чи э автомобіль, створений у 1945 році?

в) Які марки машин наявні у базі даних?

г) Знати всі машини, які випущені до 1992 року?

*Завдання 4.* Відповідно до варіанту завдання скласти програму. Додати до неї 3 факти знань, що відносяться до відповідної області. Створити власний предикат і пов’язати його з наданим предикатом. Побудувати п’ять різних запитів до бази знань.

Варіанти завдань.

1. Музика

```
predicates
songster (symbol, symbol).
clauses
songster ("sadness", "enigma").
songster ("mea culpa", "enigma").
songster ("principles of lust", "enigma").
songster ("still loving you", "scorpions").
```

2. Жанри кіно

```
predicates
movie(symbol, symbol).
Clauses
movie ("man in black", "comedy").
movie ("man in black - 2", "comedy").
movie ("King Kong", "Adventure").
```

3. Типи файлів

```
predicates
extension (symbol, symbol).
clauses
extension ("picture","*.jpg").
extension ("picture","*.tif").
extension ("picture","*.gif").
```

4. Пори року

```
predicates
season (symbol, symbol).
clauses
season ("january","winter").
season ("march","spring").
season ("april","spring").
```

5. Кількість днів у місяці

```
predicates
days (symbol, symbol).
clauses
days ("january",31).
days ("march",31).
days ("april",30).
days ("may",31).
```

6. Типи програмного забезпечення

```
predicates
class (symbol, symbol).
clauses
class ("Windows", "Operation System").
class ("DOS", "Operation System").
class ("Unix", "Operation System").
class ("Doom", "Game").
```

## ЛАБОРАТОРНА РОБОТА №2

**Тема**: Використання арифметичних операцій у Turbo Prolog

**Мета**: сформувати уміння програмування у мові Turbo Prolog з використанням арифметичних, математичних та умовних операцій.

**Пояснення** до лабораторної роботи: студенту необхідно знати теоретичний матеріал другої лекції, підготовитися до відповіді за запропонованими викладачем питаннями. Показати створені програми викладачу.

*Література*: [1], [2], [3], [4], [5].

### Питання для перевірки

1. Арифметичні оператори
2. Математичні оператори
3. Тригонометричні оператори
4. Оператори відношення та поєднання
5. Предикати введення/виведення інформації
6. Поняття уніфікації

### Короткі теоретичні відомості

*1. Арифметичні предикати і функції* Turbo Prolog

*Проста арифметика*

\+ Складання
\- Віднімання
* Множення
/ Ділення

*Цілочисельне ділення*

X mod Y Повертає залишок від ділення X на Y.
X div Y Повертає приватне від ділення X на Y. (X і Y - типу integer)

*Оператори відношення*

> Більше
< Менше
= Рівно
>= Більше або рівно
<= Менше або рівно
<> або >< Не рівно

*Логічні операції*

not НІ (вищий пріоритет)
and ТА (середній пріоритет)
or АБО (нижчий пріоритет)

*Тригонометричні функції*

Тригонометричні функції вимагають, щоб R був пов'язаний з величиною, що представляє кут у радіанах.

sin(R) Синус
cos(R) Косинус
tan(R) Тангенс
arctan(R) Арктангенс

*Арифметичні функції*

ln(R) Натуральний логарифм (R > 0)
log(R) Десятковий логарифм (R > 0)
exp(R) Експонента
sqrt(R) Квадратний корінь (R > 0)
abs(R) Абсолютна величина
round(R) округлення до цілого (-32768.5 < R < 32767.5)
trunc(R) відсікання дробової частини (-32768 < R < 32768)

*2. Предикати введення/виведення інформації*

*Введення (INPUT)*

1. readln(StringVariable) (string) – читає рядок з поточного пристрою та пов'язує її із змінною StringVariable.
2. readint(IntgVariable) (integer) – читає ціле число з поточного пристрою і пов'язує його із змінною IntgVariable.
3. readreal(RealVariable) (real) – читає дійсне число з поточного пристрою і пов'язує його із змінною RealVariable.
4. readchar(CharVariable) (char) – читає символ з поточного пристрою та пов'язує її із змінною CharVariable.
5. file_str(DosFileName,StringVariable) (string,string) – читає (записує) з файлу (у файл) DosFileName рядок і повязує її із змінною StringVariable.
6. Keypressed – перевіряє, чи натиснута деяка клавіша, не зчитуючи при цьому введений з клавіатури символ.

7. unreadchar(CharToBePushedBack) (char) – заносить символ в буфер клавіатури.
8. readterm(Domain,Variable) (DomainName, Domain) – читає терм, оголошений з ім'ям Domain. За допомогою readterm здійснюється доступ до фактів у файлі.

*Виведення (OUTPUT)*

1. write(Variable|Constant*) – записує задані значення на поточний пристрій висновку. У якості аргументу використовується список змінних і/або констант.
2. nl – перехід на новий рядок

*Форматоване виведення*

writef(FormatString,Variable|Constant*) - здійснює виведення заданих значень у вказаному форматі.

У форматі рядка використовуються наступні опції:

%d десяткове число (char і integer);
%u беззнакове ціле (char і integer);
%R покажчик на запис в базі даних (database reference number);
%X шістнадцяткове число (string, database reference numb);
%x шістнадцяткове число (char і integer);
%s рядок (symbol і string);
%c символ (char і integer);
%g дійсне число (за умовчанням для real);
%f дійсне число з фіксованою комою
\n - перехід на новий рядок
\t - табуляція

## Приклади виконаних завдань

*Приклад 1*. Скласти програму, яка буде запитувати у користувача його ім'я та привітає його

```
domains
name = symbol
predicates
  hello
clauses
  hello:-makewindow(1,7,7,"My first program",4,56,10,22),
  nl,write("Please type your name"),
```

```
    cursor(4,5), readln(Name),nl,
    write("Wellcome ",Name).
  goal
    hello.
```

Робота програми починається з виконання розділу Goal. Ціль намагається задовольнити предикат hello. Опис цього предиката є в розділі Predicates. Після цього починають послідовно виконуватися предикати, записані в правій частині предиката hello.

Після вибору опції RUN у меню Turbo Prolog, введення свого ім'я (наприклад, Ганна) у вікні введення та натиснення ENTER програма надрукує *Welcome Ганна* і буде чекати натискання клавіші пробілу (SPACE BAR).

*Приклад 2*. Обчислення куба числа, що вводитися з терміналу.

```
domains
   i=integer
predicates
    process (i)
    cube
clauses
    cube:-write("Next number,please:"), readint(X), process (X).
   process (N):- C=N*N*N, write("Cube ",N," is equal ",C,"\n"), cube.
goal
   cube.
```

*Приклад 3*. Скласти програму на знаходження мінімального

```
predicates
 min(integer, integer, integer)
 run
 clauses
 min(A, B, Min) :- A < B, Min = A, !.
 min(_, B, Min) :- Min = B.
 run:- readint(X), readint(Y),
  min(X, Y, Z),
  readint(W),
  min(Z, W, Min),
```

```
 write("Minimalnoe = ", Min),
 readchar(_).
goal
run.
```

**Завдання до виконання**

*Завдання 1.* Скласти програму обчислення середньої щільності населення країни. Повинні бути предикати площі країни та кількості населення у країні. Ввести 5 предикатів країн. Використовувати форматоване виведення інформації.

*Завдання 2.* Скласти програму обчислення функції $y=\frac{2}{x}$

*Завдання 3.* Скласти програму перевірки парності введеного з клавіатури числа.

*Завдання 4.* Розробити базу даних принців, які правили Уельсом: (родри, 844, 878), (анаравд, 878, 916), (хивел, 916, 950), (лаго, 950, 979), (хивел, 979, 985), (кадваллон, 985, 986). Сформувати такі запити до бази даних: чи був Родри принцем у 1979 році? Хто був принцем у 900 році.

*Завдання 5.* Розробити програму визначення, яке число більше. Два числа вводяться з клавіатури.

*Завдання 6.* Скласти програму обчислення функції $y=\frac{\sqrt{a-6x+4}}{2x}$

*Завдання 7.* Дані три числа a,b,c. Знайти мінімальне число серед позитивних.

## ЛАБОРАТОРНА РОБОТА №3

**Тема**: Використання рекурсії у Turbo Prolog

**Мета**: засвоїти знання про сенс рекурсії, навчитися використовувати рекурсію у програмах на Turbo Prolog.

**Пояснення** до лабораторної роботи: студенту необхідно знати теоретичний матеріал третьої та четвертої лекції та підготовитися до відповіді за запропонованими викладачем питаннями. Показати зроблені програми викладачу.

*Література*: [1], [4], [5].

### Питання для перевірки

1. Визначення рекурсії, правила використання у програмі
2. Використання віткату
3. Методи організації повторення
4. Пояснити сутність методу повторення
5. Що таке бектрекінг?
6. У чому полягає метод простої рекурсії?
7. У чому полягає метод узагальненого правила рекурсії?

### Короткі теоретичні відомості

У Turbo Prolog циклічні процеси звичайно виконуються за допомогою правил, що використовують рекурсію та відкат.

Рекурсія – визначення деякого відношення через самого себе. Правило є *рекурсивним*, якщо воно викликає саме себе у вигляді підцілі, що міститься у тілі принаймні одного з її тверджень. Оскільки в Turbo Prolog відсутні оператори циклу, то рекурсія служить основним засобом програмування циклічних процесів.

*Базис рекурсії* – це пропозиція, що визначає певну початкову ситуацію або ситуацію у момент припинення. Як правило, у цій пропозиції записується якийсь простий випадок, при якому відповідь виходить відразу навіть без використання рекурсії.

*Крок рекурсії* – це правило, у тілі якого обов’язково міститься виклик визначуваного предиката. Для того щоб уникнути зациклення, визначуваний предикат повинен викликатися не від тих же параметрів, які вказані у заголовку правила.

Вид правила, що містить рекурсію:

рекурсивне_правило:- <предикати і правила>, рекурсивне_правило.

Метод узагальненого правила рекурсії:

<им’я правила рекурсії>: -

<список предикатів>,

<предикат умови виходу>,

<список предикатів>,

<им’я правила рекурсії>,

<список предикатів>.

Будь-яке рекурсивне визначення містить принаймні одне нерекурсивне правило і одне (або декілька) правило з рекурсією.

Якщо остання умова в останньому правилі є рекурсивною, то вважається, що використовується хвостова рекурсія. Така рекурсія має перевагу перед нехвостовою рекурсією, оскільки дозволяє обмежити зростання стека та строго контролювати процес повернення. Це відбувається завдяки очищенню стека після успішного зіставлення умови, що містить рекурсію.

Прикладом рекурсивних обчислень є відомі алгоритми обчислення факторіалу та послідовності Фібоначчі, які будуть розглянуті нижче.

*Відкат* – це механізм, який Турбо-Пролог використовує для знаходження додаткових фактів і правил, необхідних при обчисленні мети, якщо поточна спроба обчислити мету виявилася невдалою. Відкат викликається невдачею у деякому місці програми, що призводить до спроби Прологу знайти наступне рішення. Відкат йде до місця, де можливо обчислити інше рішення. При цьому всі зв’язані змінні, які були зв’язані раніше до місця, де можливий інший варіант рішення, звільняються. При виклику вирішувача Прологу відкат створюється самим Прологом для видачі всіх значень рішення.

Для керування процесом відкату передбачені два вбудованих предикати: fail (невдача), який визначає, що поточний пошук рішення для деякої підцілі невдалий і потрібно почати новий пошук і пердикат cut (або !), який примусово завершує всі пошуки (відсікання).

Вид правила, що виконує повторення при відкаті:

правило_повторения:- <предикати і правила>, fail.

де fail – вбудований предикат Прологу, який завжди завершується невдачею.

Реалізація повтору методом відкату після невдачі.

місто ("Донецьк").

місто ("Макіївка").

місто ("Авдєєвка").

місто ("Дружковка").

1) Зовнішня мета 2) Внутрішня мета

? - місто (X). : - місто (X).

Відповідь: X="Донецк" Відповідь: X= "Донецк"

X=" Макіївка "

X=" Авдєєвка "

X="Дружковка"

Для однакового відображення всіх рішень слід використовувати такий синтаксис:

3) міста : - місто (X), write (X), nl, fail.

*Метод відкату і відсікання*

Cut (відсікання) – предикат, що "стирає" всі покажчики відкату. Іноді використовується символ !. Приклад, який наведений нижче, видасть на екран всі записи до "Макіївки" включно.

4) відсікання_відсікання (X) :- X = "Макіївка".

goal

міста :- місто (X), write (X), nl,

відсікання_відсікання (X), !.

На мові Turbo Prolog можуть бути реалізовані різні алгоритмічні методи, що дозволяють добитися того ж ефекту, що і ітеративні цикли у процедурних мовах. Вибір алгоритму залежить від того, як представлені дані, які потрібно обробляти. Якщо дані представлені у вигляді списку (або рекурсивної структури іншого вигляду), то оптимальним є застосування рекурсивного алгоритму. Якщо ж інформація представлена у вигляді бази даних, що містить факти, то потрібно буде використовувати алгоритм пошуку з поверненням (бектрекінг). Обидва типи циклу можна реалізувати із заданим числом повторень або потенційно нескінченним.

Рекурсивний цикл з лічильником. Цикл виконається 10 разів.

```
for(10).
for(N):– N<10,
prosess,
N1=N+1,
for(N1).
Goal for(0).
```

*Метод повторення* (МП-Метод) використовує відкат. Вид правила повтору має такий вигляд:

```
repeat.   /* повторити */
repeat :- repeat.
```

Перший repeat є твердженням, який повідомляє що предикат є repeat істинним. Перший repeat не створює підцілей, тому дане правило завжди успішне. Однак, оскільки є ще один варіант для цього правила, то покажчик відкоту встановлюється на перший repeat. Другий repeat – це правило, що використовує саме себе як компоненту (третій repeat). Другий repeat викликає третій repeat, і цей виклик обчислюється успішно, тому що перший repeat задовольняє підціль repeat. Отже, правило repeat так само завжди успішно. Предикат repeat буде обчислюватися успішно при кожній новій спробі його викликати після відкоту. Факт у правилі буде використовуватися для виконання всіх підцілей програми. Таким чином, repeat це рекурсивне правило, що ніколи не буває невдалим.

З використанням предиката repeat можна також організувати будь-яку відповідну умову закінчення роботи циклу:

```
predicates
run.
repeat.
clauses
repeat.
repeat :- repeat.
run:- repeat,
write("Prodovgity vvedennya? (y/n)" ),
readchar(Ans), Ans = 'n'.
```

*Бектрекінг (алгоритм пошуку з поверненням*) – механізм повернення, що здійснює відкат програми до тої крапки, у якій вибирався уніфікований з останньою підцілью діз'юнкт (елемент предиката). Для цього крапка, де вибирався один з можливих уніфікованих з підцілью діз'юнктів, запам'ятовується у спеціальному стеці, для наступного повернення до неї та вибору альтернативи у випадку невдачі. При відкоті всі змінні, які були зазначені у результаті уніфікації після цієї крапки, знову стають вільними.

У підсумку виконання програми може завершитися невдачею, якщо одну з підцілей не вдалося уніфікувати з жодним диз'юнктом програми,

і може завершитися успішно, якщо був виведений порожній діз'юнкт, а може й просто зациклитися.

Turbo Prolog має убудований механізм логічного висновку, завдяки чому від користувача потрібно тільки опис свого завдання за допомогою апарата логіки предикатів першого порядку, а пошук рішення система бере на себе. Розглянемо приклад використання бектрекінга в Turbo Prolog:

```
p :- q, s.
p :- r, t.
```

Цю програму можна прочитати у такий спосіб. При виконанні процедури *p* виконуються процедури *q* і *s*. Якщо вони завершуються успішно, то й процедура *p* уважається успішно завершеною. Якщо це не так, то виконуються процедури *r* і *t*. У цьому випадку процедура успішно завершується, якщо *r* і *t* завершені успішно. У противному випадку процедура *p* зазнає невдачі. Цей процес повернення до пошуку інших рішень називається *бектрекінгом (backtracking)*.

Принципова різниця між backtracking і рекурсією полягає у тому, що перший не дозволяє передавати дані між кроками ітерацій, тому що кожного разу при досягненні мети всі змінні "звільняються".

## Приклади виконаних завдань

*Приклад 1.* Розглянемо використання методу узагальненого правила рекурсії (ОПР) проілюструємо на прикладі програми генерації ряду чисел.

```
predicates
write_number(integer).
 clauses
   write_number(Number) :-
      Number < 8,
      write(Number), nl,
      Next_Number = Number + 1,
      write_number(Next_number).
goal
write_number(1).
```

Програма починається зі спроби обчислити підціль write_number(1). Спочатку програма зіставляє підціль з головою правила write_number(Number). Цього разу зіставлення успішно внаслідок того, що змінної Number привласнено значення 1. Програма порівнює це

значення з 8; ця умова виходу. Тому що 1 менше 8, то підправило успішно. Наступний предикат видає значення, привласнене Number. Змінна Next_Number одержує значення 2, а значення Number збільшується 1.

*Приклад 2.* Скласти програму яка запитує введення символів та виводить їх на екран. Вихід за допомогою слова exit. Використати метод повторення.

```
predicates
luna.
repeat.
umova_ostanova (symbol)
clauses
repeat.
repeat :- repeat.
luna  :- repeat,
readln (X),write (X), nl,
umova_ostanova (X), !.
umova_ostanova (exit):- nl, write ("GoodBy").
```

*Приклад 3.* Скласти програму для *о*бчислення факторіалу 4!.

```
domains
  n, f = real
predicates
  factorial (n,f)
clauses
  factorial (1, 1).
  factorial (N, F):- N>0, N1=N-1, factorial (N1, F1), F=F1*N.
goal
factorial (4,X), write (X).
```

При кожному виклику диз'юнкта factorial генеруються нові змінні, які діють завжди тільки на своєму рівні вкладеності, поки не зустрінеться умова припинення обчислень. Тільки після цього по дорозі назад проходження рекурсії визначаються результати, які передаються вгору.

*Приклад 4.* Обчислення числа Фібоначчі. Числа Фібоначчі або Послідовність Фібоначчі – числова послідовність, що володіє поряд властивостей. За визначенням, перші дві цифри в послідовності Фібоначчі 0 і 1 (або альтернативно, 1 і 1). Сума двох сусідніх чисел послідовності дає значення наступного за ними (наприклад, 1+1=2; 2+3=5 і т.д.), що підтверджує існування так званих коефіцієнтів Фібоначчі, тобто постійних співвідношень.

```
fib(1,1):-!. /* перше число Фібоначчі рівне одиниці */
fib(2,1):-!. /* друге число Фібоначчі рівне одиниці */
fib(N,F):-
N1=N-1, fib(N1,F1), /* F1 це N-1-е число Фібоначчі */
N2=N-2, fib(N2,F2), /* F2 це N-2-е число Фібоначчі */
F=F1+F2. /* N-е число Фібоначчі рівне сумі N-1 і N-2 */
```

*Приклад 5*. Скласти програму обчислення суми чисел від 1 до N включно. Створити предикат вигляду sum(N,X).

```
predicates
domains
  n, f = real
predicates
  sum (n,f)
clauses
  sum (1, 1).
  sum (N, F):- N>0, N1=N-1, sum (N1, F1), F=F1+N.
goal
sum(3,X),write(X)
```

*Приклад 6.* Нескінченний рекурсивний цикл. Виконується введення і друк символів. Процес повторюється до натиснення клавіші ENTER, що має кодове число 13.

```
predicates
typewr(char).
clauses
typewr('\13').
typewr(Char):- write(Char),
readchar(C), typewr(C).
Goal
```

readchar(Char), typewr(Char).

**Завдання до виконання**

*Завдання 1.* Скласти програму обчислення суми парних чисел від 1 до N. Створити предикат вигляду sum(N,X).

*Завдання 2*. Скласти програму обчислення суми додатніх чисел від 1 до N. Змінна N вводиться з клавіатури. Створити предикат вигляду sum(N,X).

*Завдання 3.* Скласти програму табулювання функції y = 3x-2, де x – число від 2 до 5.

*Завдання 4*. Скласти програму, яка підраховує суму грошей, якщо клієнт поклав у банк K грн. під N відсотків на Z років. Значення змінних задаються з клавіатури.

*Завдання 5*. Скласти програму, яка виводить на екран всі парні числа від 53 до 62 у зворотньому порядку.

*Завдання 6*. Створіть програму, що обчислює по натуральному числу N, введеному з клавіатури, суму і число непарних чисел, що не перевершують N.

*Завдання 7*. Створіть програму, що обчислює найбільший спільний дільник двох натуральних чисел.

## ЛАБОРАТОРНА РОБОТА №4

**Тема**: Операції над списками у Turbo Prolog

**Мета**: розвинути вміння використовувати списки для обробки інформації різних типів, розвинути уміння основні принципи логічного програмування.

**Пояснення** до лабораторної роботи: студенту необхідно знати теоретичний матеріал третьої лекції та підготовитися до відповіді за запропонованими викладачем питаннями. Показати виконані завдання викладачу.

*Література*: [1], [4], [5], [6], [7].

### Питання для перевірки

1) Що таке список у Turbo Prolog?
2) Які операції над списками
3) Описати структуру списку?
4) Як розділяються елементи списку?
5) Назвати приклади списків
6) Назвати предикати для роботи зі списками
7) Друк списків

### Короткі теоретичні відомості

*1. Оголошення списку*

Список – досить широко використовувана структура даних у області числового програмування. Список є основною структурою даних в Пролозі.

Список – це впорядкована послідовність елементів, яка може мати довільну довжину. Ознака впорядкований указує на те, що порядок елементів в послідовності є істотним.

Елементами списку можуть бути будь-які терми – константи, змінні, структури, які включають, звичайно, і інші списки.

У загальному випадку голова списку може бути будь-яким об’єктом мови Пролог, а хвіст обов'язково повинен бути списком. Оскільки хвіст є список, то він або порожній, або має свої власну голову і хвіст. Елементи списку розділяються комами та заключаються у квадратні дужки. Будь-який список є:

– або порожній список (атом []);
– або непорожній список – структура, що складається з двох частин:

– перший елемент – голова (Head) списку;
– другий елемент – хвіст (Tail) списку.

Елементи списку можуть бути будь-якими, у тому числі і складеними об’єктами. Зокрема, елементи списку самі можуть бути списками.

listI = integer* /* список, елементи якого цілі числа */
listR = real* /* список, що складається з речовинних чисел */
listC = char* /* список символів */
lists = string* /* список, що складається з рядків */
listL = listI* /* список, елементами якого є списки цілих чисел */

Для підвищення наочності програм в Пролозі передбачаються спеціальні засоби для спискової нотації, що дозволяють представляти списки у вигляді [Елемент1, Елемент2...] або [Голова | Хвіст ] або [Елемент1, Елемент2... | Останній].

Тут знак | використовується для відділення початку списку від кінця.

Списки можуть містити інші списки або змінні. Наприклад, в Пролозі допустимі наступні списки:

– []
– [конкретна, людина [любить, ловити, рибу]]
– [a, V1, b [X, Y]]

*2. Операції над списками*

*Приналежність до списку (member)*

Відношення приналежності записується у вигляді двох пропозицій:

member(X,[X|Tail]).
member(X,[_|Tail]):- member(X,Tail).

Ця форма запису заснована на наступних міркуваннях: або X - голова списку, або X належить хвосту цього списку. Приналежність до списку можна записати іншим виглядом: member (X,[X |_]).

Цей запис констатує, що X є елементом списку, який має X у якості голови. Відмітимо, що ми використовували анонімну змінну '_' для позначення хвоста списку. Це зроблено тому, що ми ніяк не використовуємо хвіст списку в цьому конкретному факті.

Друге правило говорить про те, що X належить списку за умови, що він входить в хвіст цього списку, що позначається через Tail. І немає кращого шляху, ніж використовувати той же самий предикат “належить” для того, щоб визначити, чи належить X хвосту списку. У цьому і полягає суть рекурсії. На Пролозі це виглядає так:

member (X,[_ | Tail]):- member (X, Tail).

Правило констатує, що X є елементом списку, якщо X є елементом хвоста цього списку. Відмітимо, що у даному прикладі використовується анонімна змінна '_', оскільки нас не цікавить ім'я змінної, що позначає голову списку.

*З'єднання (конкатенація) списків (conc)*

Позначається через conc(L1,L2,L3).

Тут L1 і L2 - два списки, L3 - список, що отримується при їх зчепленні. Визначення відношення conc містить два випадки:

- якщо перший аргумент - порожній, то другий і третій аргументи є одним і тим же списком: conc([],L,L).
- якщо перший аргумент відношення conc не порожній, то він має голову і хвіст - [X|L1]. Результатом зчеплення є список [X|L3], де L3 - отриманий після зчеплення списків L1 і L2: conc([X|L1],L2,[X|L3]):- conc(L1,L2,L3).

Програму для conc можна застосувати "у зворотному напрямі" - для розбиття списку на дві частини. Наприклад: Goal: conc(L1,L2, [a,b,c]).

L1=[]
L2=[a,b,c]
L1=[a]
L2=[b,c]
L1=[a,b]
L2=[c]
L1=[a,b,c]
L2=[]

Використовуючи conc можна визначити відношення приналежності наступним еквівалентним способом: member(X,L):- conc(_,[X|_],L).

Тут символом "_" позначені анонімні змінні (змінні, що зустрічаються в пропозиції тільки по одному разу).

Додавання елементу (append)

Додати елемент до списку - це вставити його в самий початок так, щоб він став новою головою. Якщо X - це новий елемент, який додають до списку L, то предикат додавання елемента в список можна записати таким чином:

```
add(X,[], [X]).
add(X,L, [X|L]) :- !.
```

Видалення елементу (remove)

Маємо два випадки:

- якщо X - голова списку, то результат видалення - хвіст списку;
- якщо X - в хвості списку, то його потрібно видалити звідти.

В результаті відношення remove визначається так:

```
remove(X, [X|Tail], Tail).
remove(X, [Y|Tail], [Y|Tail1]):- remove(X,Tail,Tail1).
```

Якщо в списку зустрічається декілька входжень елементу X, то remove зможе виключити їх всіх за допомогою повернень.

Видалення зі списку повторюваних елементів

Аргументами предиката unik є два списки: вихідний і результуючий.

```
unik([H|T],T1):-member(H,T), unik (T,T1).
unik ([H|T],[H|T1]):-not(member(H,T)), unik(T,T1).
```

**Приклади виконаних завдань**

*Приклад 1.* Створити предикат належності списку

```
domains
  s=symbol
  s_list=s*
predicates
member (s,s_list).
clauses
  member(X,[X|_]).
  member(X,[_|Y]):- member(X,Y).
? member (d,[a,b,c,d,e,f,g]).
Yes
? member (d,[a,b,c,e,f,g]).
No
```

*Приклад 2.* Створити програму для об’єднання списків

```
domains
  L =symbol*
predicates
```

```
conc (L,L,L).
clauses
conc([],L,L).
conc([X|L1],L2,[X|L3]):- conc(L1,L2,L3).
  Goal
  conc([a,b,c],[x,y,z],L).
```

*Приклад 3.* Створити програму для приєднання елемента у кінець списку

```
domains
  l=integer*
predicates
  add(integer,l,l)
clauses
  add(E,[],[E]).
  add(E,[H|T],[H|T1]):-add(E,T,T1).
goal
  write("vvedite element "),
  readint(E),
  add(E,[1,2,3],L),
  write("L=",L).
```

*Приклад 4.* Визначення суми елементів списку

```
domains
  i=integer
  i_list=i*
predicates
  sum_list(i_list,i)
clauses
  /* Якщо список порожній, то сума дорівнює нулю */
  sum_list([],0).
  /* Інакше знайти суму елементів хвоста списку
      і додати до них голову */
  sum_list([H|T],Sum):-sum_list(T,Sum1), Sum=H+Sum1.
goal
sum_list([2,2,2,1],X), write (X).
```

*Приклад 5.* Виведення списків

```
domains
    i_list=integer*
    n_list=symbol*
predicates
    writelist(i_list)
    writelist(n_list)
clauses
    writelist([]).
    writelist([H|T]):- write(H," "),writelist(T).
goal
writelist ([1,2,3]).
```

**Завдання до виконання**

*Завдання 1.* Скласти програму, яка знаходить мінімальне число у списку та виводить його на екран.

*Завдання 2.* Скласти програму, яка зчитує цілі числа з терміналу і заносить їх у список. Після набору кожного цілого числа потрібно натиснути ENTER.

*Завдання 3.* Скласти програму, яка оголошує список із семи значень та виводить на екран тільки значення 1,2,4 елементів.

*Завдання 4.* Заданий список. Вивести списки з трьох непарних елементів, не обов’язково на перших місцях.

*Завдання 5.* Заданий список. Обчислити кількість парних елементів

*Завдання 6.* Створити програму, яка продублює входження кожного елементу списку, наприклад було [1,2,3], а стало [1,1,2,2,3,3].

## ЛАБОРАТОРНА РОБОТА №5

**Тема**: Обробка рядків у Turbo Prolog

**Мета**: розвинути вміння створювати програми на Turbo Prolog, які обробляють рядкову інформацію.

**Пояснення** до лабораторної роботи: студенту необхідно знати теоретичний матеріал шостої лекції та підготовитися до відповіді за запропонованими викладачем питаннями. Показати виконані завдання викладачу.

*Література*: [1], [4], [5].

### Питання для перевірки

1. Предикат str_len
2. Предикат concat
3. Предикат frontchar
4. Предикат frontstr, fronttoken

### Короткі теоретичні відомості

Під рядком в Пролозі розуміється послідовність символів, взята у подвійні лапки.

Стандартні предикати обробки рядків в Turbo Prolog:

1. concat(Str1,Str2,Str1_2) – предикат, який утворює рядок Str1_2 – це об'єднання рядків Str1 і Str2; при цьому принаймні два параметри повинні бути визначені заздалегідь.

2. frontchar(Str1_2,Char,Str2) - працює відповідно до рівняння:

Str1_2 = Char U Str2

3. frontstr(Length,InpString,StartString,RestString) – пееносить перші Length символів рядка InpString в рядок StartString, останні – у рядок RestString.

4.fronttoken(String,Token,RestString), де String - вхідний рядок;
Token - перший атом рядка (послідовність символів до пропуску);
RestString - залишок рядка.

5. Str_len (String,Length) – визначає довжину рядка.

Є три варіанти використання функції Str_len. Перший, найбільш природний варіант використання цього предиката, коли перший аргумент зв'язаний, а другий вільний. В цьому випадку в другий аргумент буде поміщена кількість символів в першому аргументі.

Другий, також очікуваний варіант, коли обидва аргументи зв’язані. В цьому випадку предикат буде успішний, якщо довжина першого аргументу співпадатиме з другим аргументом, і неуспішний інакше.

Третій, не такий поширений варіант використання, коли другий аргумент зв’язаний, а перший – вільний. У цій ситуації перший аргумент буде зазначений рядком, що складається з пропусків, причому кількість пропусків буде рівна другому аргументу.

**Приклади виконаних завдань**

*Приклад 1.* Перетворення рядка в список символів

```
domains
l = char*
predicates
list_str(l,string)
clauses
list_str([],""). /* порожньому рядку відповідає
       порожній список */
list_str([H|T],S):- list_str(T,S1),
        /* S1 — рядок, утворений
         елементами списку T */
     frontchar(S,H,S1).
        /* S — рядок, одержаний
         дописуванням рядка S1
         до першого елементу списку H */
goal
list_str([63,62,1],S),write(S)
```

*Приклад 2.* Скласти програму, яка створює рядок, що містить перші букви слів вхідного рядка

```
domains
   s=string

predicates
   preobr(s, s, s)
```

```
clauses
    preobr(Stroka, Nakopitel, Resyltat):- frontToken(Stroka, Slovo,
OstatokStroki),
            frontChar(Slovo, PervayaBykva, _),      !,
            str_char(PervayaBykva1, PervayaBykva),
            concat (Nakopitel, PervayaBykva1, Nakopitel1),
            preobr(OstatokStroki, Nakopitel1, Resyltat).
    preobr(_, Resyltat, Resyltat).
Goal
Preobr (“aa bb ff”, ” “, S), write (S).
```

*Приклад 3.* Скласти програму, яка видаляє із рядка заданий набір символів та виводить отриманий рядок на екран

```
domains
s=string
predicates
deleteSlovo(s,s,s)
clauses
deleteSlovo(Slovo,Stroka,Result):- concat(Slovo,D,Stroka), !,
deleteSlovo(Slovo,D,Result).
deleteSlovo(Slovo,Stroka,Result):- frontstr(1,Stroka,D,E), !,
deleteSlovo(Slovo,E,F),
concat(D,F,Result).
deleteSlovo(_,_,"").
goal
deleteSlovo("aa", "saba aaba aba tgt", X), write(X).
```

*Приклад 4.* Обчислення середнього віку

```
domains
  name, address = string
  age = integer
  list = age*
predicates
  person(name, address, age)
  sumlist(list, age, integer)
goal
```

```
    findall(Age, person(_, _, Age), L), sumlist(L, Sum, N),  Ave = Sum/N,
write("Average =", Ave), nl.
  clauses
    sumlist([], 0, 0).
    sumlist([H|T], Sum, N) :- sumlist(T, S1, N1), Sum=H+S1, N=1+N1.
    person("Sherlock Holmes", "22B Baker Street", 42).
    person("Pete Spiers", "Apt. 22, 21st Street", 36).
    person("Mary Darrow", "Suite 2, Omega Home", 51).
```

В даному випадку використовується вбудований предикат findall(Age, person(_, _, Age), L), де:

- Value – об'єкт вхідного предиката (що вибирається);
- person(_, _, Age) – вхідний предикат (звідки вибирається);
- L - вихідний список (куди вибирати), його елементи належать тому ж домену, що й об'єкт вхідного предиката.

*Приклад 5.* Створити предикат, який по рядку і символу підрахує кількість входжень цього символу в даний рядок. Предикат матиме три аргументи: перші два – вхідні (рядок і символ), третій – вихідний (кількість входжень другого аргументу в перший).

Рішення буде рекурсивним. Рекурсія по рядку, в якому ведеться підрахунок кількості входжень даного символу. Якщо рядок порожній, то не важливо, входження якого символу ми рахуємо, все одно відповіддю буде нуль. Це базис. Кроків рекурсії буде два залежно від того, чи буде першим символом рядка символ, входження якого ми рахуємо, чи ні. У першому випадку потрібно підрахувати, скільки разів шуканий символ зустрічається в залишку рядка, і збільшити одержане число на одиницю. У другому випадку (коли перший символ рядка відмінний від символу, який ми рахуємо) збільшувати одержане число не потрібно. При розщеплюванні рядка на перший символ і хвіст потрібно скористатися вже знайомим нам предикатом frontchar.

```
predicates
    char_count(string,char,integer)
clauses
    char_count("",_,0). /* Будь-який символ не зустрічається в
порожньому рядку ні разу*/
    char_count(S,C,N):- frontchar(S,C,S1),!,
```

```
    /* символ C виявився першим символом рядка S, в S1 – символи
рядка S, що залишилися */
          char_count(S1,C,N1),
            /* N1 – кількість входжень символу C в рядок S1 */
          N=N1+1.
   /* N – кількість входжень символу C в рядок S виходить з кількості
входжень символу C в рядок S1 додаванням одиниці */
   char_count(S,C,N):- frontchar(S,_,S1),
   /* першим символом рядка S виявився символ, відмінний
    від початкового символу C, в S1 – символи рядка S, що залишилися
*/
          char_count(S1,C,N).
            /* в цьому випадку кількість входжень символу C в рядок S
співпадає з кількістю входжень символу C в рядок S1 */
   goal
   char_count("aaddffa",'a',N),write(N)
```

**Завдання до виконання**

*Завдання 1.* Написати програму, що у будь-якому заданому з клавіатури тексті визначає кількість заданих з клавіатури літер.

*Завдання 2.* Написати програму, що в заданому з клавіатури реченні підраховує кількість слів.

*Завдання 3.* Написати програму, яка у заданому з клавіатури виразі перевіряє вірність розставлених дужок (кількість відкриваючих повинна дорівнювати кількості закриваючих).

*Завдання 4.* Підрахувати скільки разів серед символів заданого рядка зустрічається буква “F”.

*Завдання 5.* Створіть предикат, який видалятиме з даного рядка всі входження заданого символу

*Завдання 6.* Створить предикат, який продублює входження кожного символу в рядок

## ЛАБОРАТОРНА РОБОТА №6

**Тема**: Введення та виведення даних у Turbo Prolog

**Мета**: сформувати у студентів вміння створювати програми на Turbo Prolog для занесення інформації до файлу та зчитування її із файлів.

**Пояснення** до лабораторної роботи: студенту необхідно знати теоретичний матеріал п’ятої лекції та підготовитися до відповіді за запропонованими викладачем питаннями. Показати виконані завдання викладачу.

*Література*: [1], [4], [5].

### Питання для перевірки

1. Взаємодія з файлами
2. Форматний вивід writef
3. Уведення й виведення чисел і символів
4. Убудований предикат findall

### Короткі теоретичні відомості

*1. Взаємодія з файлами*

Введення і висновок в Пролозі організовується за допомогою спеціальних предикатів читання і записи, які можуть розглядатися як аналоги відповідних підпрограм в мовах Паскаль і Сі.

У загальному випадку програма, яка написана на Turbo Prolog, взаємодіє з декількома файлами (зокрема з "псевдофайлом" keyboard (клавіатура) і "псевдофайлом" screen (екран)). Вона прочитує дані з декількох вхідних файлів, званих вхідними потоками, і виводить дані в декілька вихідних файлів, званих вихідними потоками.

У кожен момент виконання програми лише два файли є "активними": один для введення, інший для виведення. У початковий момент ці два потоки відповідають терміналу. Поточний вхідний потік може бути замінений на інший файл name_of_file за допомогою мети:

readdevice(name_of_file).

Така мета завжди успішна (якщо тільки з файлом name_of_file все гаразд), а як побічний ефект відбувається перемикання введення з попереднього вхідного потоку на файл name_of_file.

*Використання предиката readdevice*

Приведена нижче послідовність цілей прочитує інформацію з файлу myfile, а потім перемикає введення назад на термінал.

```
...
openread(myfile,"myfile.txt")
/* файл myfile відкривається для читання */
readdevice(myfile)
/* стандартний вхідний потік зв'язується з файлом myfile */
filepos(myfile,X,0)
/* поточний покажчик встановлюється на позицію X, яка
відлічується від початку файлу myfile */
readchar(Y)
/* з позиції X читається символ і призначається змінною Y */
readdevice(keyboard)
/* стандартний вхідний потік зв'язується з клавіатурою */
```

Поточний вихідний потік може бути змінений за допомогою мети вигляду writedevice(name_of_file).

*Використання предиката writedevice.*

```
openwrite(outfile,"outfile.txt")
/* файл outfile відкривається для запису */
writedevice(outfile)
/* стандартний вихідний потік зв'язується з файлом outfile */
write(X)
closefile(outfile)
/* закриття файлу outfile */
writedevice(screen)
/*стандартний вихідний потік зв'язується з екраном дисплея */
```

Окрім вищезазначених використовуються наступні стандартні предикати роботи з файлами:

1) *openappend* (логічне ім’я файлу, фізичне ім’я файлу) – подібний openread, але файл відкривається для доповнення;

2) *openmodify* (логічне ім’я файлу, фізичне ім’я файлу) – подібний openread, але файл відкривається для модифікації (читання і запис);

3) *filepos* (логічне ім’я файлу, позиція, режим) – встановлює поточний покажчик на задану позицію у вказаному файлі. Об’єкт, вказуючий на позицію, повинен бути речового типу (real), а режим –

цілого типу (integer). Якщо режим рівний 0, то позиція відлічується від початку файлу; якщо 1, то - від поточного значення покажчика; якщо 2, то - від кінця файлу. Цей предикат використовується для прямого доступу до файлу.

4) *readchar* (Var) - прочитує символ з вхідного потоку ( за умовчанням з клавіатури) і привласнює його змінній Var;

5) *readint* (Var) - считывает целое число и присваивает его переменной Var;

6) *readln* (Var) - прочитує символи з вхідного потоку до натиснення клавіші Enter. Введені символи привласнюються змінною Var, яка повинна бути строковою (string) або символьною (symbol).

7) *readreal* (Var) – прочитує дійсне число;

8) *write* (Arg1, Arg2 ...) - виводить значення аргументів на поточний пристрій (за умовчанням, на екран дисплея). Аргументи Arg1, Arg2 ... можуть бути константами або змінними, яким заздалегідь привласнені необхідні значення.

9) nl – викликає переведення каретки на наступний рядок.

У предикаті write можна використовувати символи, що починаються із знаку \. Вони мають спеціальні значення:

\до - символи, ASCII, що мають, код числа до;

\n - повернення каретки і переклад рядка;

\t - табуляція.

Файли можуть оброблятися тільки послідовно.

Кожен запит на читання з вхідного файлу приводить до читання в поточній позиції поточного вхідного потоку. Після цього поточна позиція буде переміщена на наступний, ще не прочитаний елемент даних. Наступний запит на читання приведе до зчитування, починаючи з цієї нової поточної позиції.

Запис проводиться так само: кожен запит на виведення інформації приведе до того, що вона буде приєднана до кінця поточного вихідного потоку.

**Приклади виконаних завдань**

*Приклад 1.* Запис символів у файл myfile.f, який створюється на поточному диску. Введення символів відбувається до введення символу «#».

```
domains
```

```
file=myfile
predicates
read_in_loop
clauses
read_in_loop:- readchar(X), X<>'#13',!,write (X), read_in_loop.
goal
openwrite (myfile,"myfile.txt"), writedevice (myfile),
not(read_in_loop),
closefile(myfile),writedevice(screen),
write("\n File zapisan").
```

*Приклад 2.* Читання символу з файлу і виведення його на екран

```
domains
 file=infile
 predicates
 position
 clauses
 position:- readdevice(keyboard), nl,write("vvedite poziciuy "),
 readreal(X), readdevice(infile), filepos(infile,X,0),
 readchar(Y), write(" Tyt zapisaniy simvol: ",Y), position.

 goal
 write(" Vvedite nazvu faylu \n "),
 readln(Fname), openread(infile,Fname), position.
```

*Приклад 3.* Скласти програму, яка записує результат складання чисел, які містяться у перших рядках файлів, у третій файл.

```
domains
file=file_in;file_out
i=integer
 predicates
load_from_file(i)
run
 Clauses
load_from_file(A):-readint(A).
run :-
openread(file_in,"Slag1.txt"),readdevice(file_in),
```

```
    load_from_file(A), closefile(file_in),
    openread(file_in,"Slag2.txt"),readdevice(file_in),
    load_from_file(B), closefile(file_in),
    C=A+B,
    openwrite(file_out,"result.txt"),writedevice(file_out),write(C),
closefile(file_out),write(C), readchar(_).
    goal
run
```

**Завдання до виконання**

*Завдання 1.* Скласти програму, яка відкриває файл 1.txt, розташовує всі рядки задом наперед та заносить весь текст у файл 2.txt.

*Завдання 2.* Зчитати з файлу символи, перевести їх у список та вивести на екран.

*Завдання 3.* Протабулювати функцію y = x – 4, де x – значення від –3 до 5. Отримані значення занести файл "myfile.txt".

*Завдання 4.* Скласти програму, яка підраховує кількість слів у файлі та результат виводить на екран.

Завдання 5. Символьний файл містить різні букви латинського алфавіту. Скласти програму, створює файл з попереднього файлу, але без символів “a”.

## ЛАБОРАТОРНА РОБОТА №7

**Тема**: Бази даних у Турбо-Пролозі

**Мета**: розвинути вміння створювати програми на Пролозі, які використовують логічні бази даних.

**Пояснення** до практичної роботи: студенту необхідно знати теоретичний матеріал сьомої лекції та підготовитися до відповіді за запропонованими викладачем питаннями. Показати зроблені програми викладачу.

*Література*: [1], [4], [5].

### Питання для перевірки

1. Звертання до бази даних
2. Вбудовані предикати для роботи з даними
3. Вбудовані предикати для роботи з базами даних
4. Накопичення в базі даних відповідей на питання

### Короткі теоретичні відомості

У Турбо-Пролозі є спеціальні засоби для організації баз даних. Ці засоби розраховані на роботу з реляційними базами даних, тому що Турбо-Пролог особливо гарний для написання діалогової системи саме для реляційної БД: внутрішні уніфікаційні процедури мови здійснюють автоматичну вибірку фактів з потрібними значеннями відомих параметрів і привласнюють значення ще не певним. До того ж механізм відкоту дозволяє знаходити всі наявні відповіді на зроблений запит.

Розділ database у Турбо-Пролозі призначений для опису предикатів бази даних, таких як dplayer. Всі різні твердження цього предиката становлять динамічну базу даних Турбо-Прологу. База даних називається динамічної, так під час роботи програми з її можна видаляти будь-які твердження, що втримуються в ній, а також додавати нові.

У цьому полягає її відмінність від "статичних" баз даних, де твердження є частиною коду програми й не можуть бути змінені під час рахунку. Інша важлива особливість динамічної бази даних полягає в тому, що така база може бути записана на диск, а також лічена з диска в оперативну пам'ять.

*1. Звертання до БД*

Щоб зрозуміти, як у Турбо-Пролозі реалізується звертання до БД, розглянемо запит створюваної БД - „футбольная команда”

```
dplayer("Bernie Kosar",Team,Pos).
```

У цьому твердженні Team і Pos є змінні, значення яких потрібно знайти. Коли цей запит (ціль) випробовується, процедури Турбо-Прологу переглядають твердження БД на предмет зіставлення із твердженням, що містить Bernie Kosar.

Тому що в базі даних таке твердження присутнє, то змінної Team привласнюється значення Cleveland Browns, а змінної Pos - QB (виходячи із БД наведеного нижче приклада). Якщо трактувати dplayer як предикат БД Турбо-Пролог, то звідси треба його опис

```
database
  dplayer(name,team,position)
```

Розділ database у Турбо-Пролозі призначений для опису предикатів бази даних, таких як dplayer. Всі різні твердження цього предиката становлять динамічну базу даних Турбо-Прологу. База даних називається динамічної, так під час роботи програми з її можна видаляти будь-які твердження, що втримуються в ній, а також додавати нові.

У цьому полягає її відмінність від "статичних" баз даних, де твердження є частиною коду програми й не можуть бути змінені під час рахунку. Інша особливість динамічної бази даних полягає в тому, що така база може бути записана на диск, а також лічена з диска в оперативну пам’ять.

Переважна частина інформації бази даних представлена у вигляді тверджень статичної БД. Ці дані заносяться в динамічну БД відразу послу активізації програми. Для цього використовуються предикати asserta і assertz, які розглянуті нижче. Загалом, предикати статичної БД мають інше ім'я, але ту ж саму форму подання даних, що й предикати динамічної. Предикат статичної БД, що відповідає предикату dplayer динамічної бази даних, тобто БД «футбольна команда» має вигляд

```
predicates
  player(name,team,position)
clauses
  player("Dan Marino","Miami Dolphins","QB").
  player("Richart Dent","Chicago Bears","DE").
```

```
player("Bernie Kosar","Cleveland Browns","QB").
player("Doug Cosbie","Dallas Cowboy","TE").
player("Mark Malone","Pittsburgh Steelers","QB").
```

Помітимо, що вся відмінність предиката dplayer у порівнянні з player полягає лише в одній зайвій букві терма. Додавання латинської букви d - звичайний спосіб розрізняти предикати динамічної й статичної баз даних.

Правилом для занесення в динамічну БД інформації із тверджень предиката player служить

```
assert_database :- player(Name,Team,Number),
 assertz( dplayer(Name,Team,Number) ),
 fail.
assert_database :- !.
```

У цьому правилі застосовується метод відкоту після невдачі, що дозволяє перебрати всі твердження предиката player (гравці).

*2.Вбудовані предикати для роботи з даними*

Пролог-програму можна розглядати як реляційну базу даних, тобто опис деякої безлічі відносин. Опис відносин присутній або в явному вигляді (факти), або в неявному вигляді (правила).

Вбудовані предикати дають можливість коректувати цю базу даних в процесі виконання програми. Це робиться:

1) додаванням до програми (в процесі обчислень) нових фактів;

2) викреслюванням з неї вже існуючих фактів.

Наступні цілі (предикати) виконують операції над БД:

1.Цель assert(d) завжди успішна і додає факт d до бази даних;

2.Цель retract(d) видаляє факт, зіставний з d;

3.asserta(d) - забезпечує запис в базу даних нового факту перед наявними фактами для заданого відношення;

4.assertz(d) - забезпечує запис в базу даних нового факту після всіх наявних фактів для заданого відношення.

Оголошення динамічної бази даних, в яку факти можуть додаватися під час виконання програми або вибиратися з файлу за допомогою предиката consult, здійснюється за допомогою ключового слова database.

*3. Вбудовані предикати для роботи з базами даних*

*consult(Dos_File_Name).* Додає текстовий файл із ім'ям Dos_File_Name до поточної бази даних. Текстовий файл може бути, наприклад, створений у результаті виконання предиката save. Цей файл містить факти, які повинні бути описані в розділі DbaseDom. Виконання предиката не буде успішним, якщо у файлі є синтаксичні помилки.

Предикат consult неуспішний, якщо файл із зазначеним ім'ям відсутній на диску, або якщо цей файл містить помилки, як, наприклад, у випадку невідповідності синтаксису предиката з файлу описам з розділу програми database, або якщо вміст файлу неможливо розмістити в пам'яті через відсутність місця.

*save(Dos_File_Name).* Записує динамічну базу даних на диск у текстовий файл із ім'ям Dos_File_Name. Після цього файл можна знову завантажити в оперативну пам'ять, використовуючи предикат consult. Для зберігання кожного факту використовується окремий рядок. Текстовий файл, що представляє собою всю базу даних, можна переглянути й змінити, використовуючи редактор Прологу.

*4. Накопичення в базі даних відповідей на питання*

Одним з корисних застосувань предиката asserta є накопичення вже обчислених відповідей на питання. Хай, наприклад, в програмі визначений предикат solve(Problem, Solution).

Користувач може поставити питання і зажадати, щоб відповідь на нього була збережена у базі даних. Це дозволить полегшити отримання відповідей на майбутні питання:

solve(Problem,Solution), asserta(solve(Problem,Solution)).

Якщо в першій з приведених цілей буде успіх, відповідь (Solution) буде збережена, а потім використаний так само, як і будь-який інший додаток, при відповіді на подальші питання. Перевага такого “запам’ятовування” полягає в тому, що на подальші питання, які зіставляються з доданим фактом, відповідь буде отримана, як правило, значно швидше, ніж вперше. Відповідь тепер буде отримана як факт, а не як результат обчислень, що вимагають, можливо, тривалого часу.

## Приклади виконаних завдань

*Приклад 1. Створити базу даних*

```
  domains
tip, fun=symbol
x,kol=integer
database
ms(tip,fun,x,kol)
/* tip - тип мікросхеми; fun - функція, що реалізовується;
 x  - число входів;  kol - число елементів */
  clauses
ms(k155la3,i_ne,2,4).
ms(k155la4,i_ne,3,3).
ms(k155la1,i_ne,4,2).
ms(k155ln1,ne,1,6).
ms(k155le1,ili,2,4).
ms(k155li3,i,3,3).
```

Введення рядка assertz(ms(k155ir1,rg,4,4)) призведе до додавання факту ms(k155ir1,rg,4,4) в базу даних.

Введення рядка retract(ms(Q,i_ne,W,E)) призведе до видалення з БД всіх об'єктів для елементів типу i_ne.

*Приклад 2.* Простий ітераційний процес. Дана програма змінює значення лічильника counter(i) в базі даних від i=0 до i=100.

```
      database
       counter(integer)
      predicates
       repeat
       count
      goal
       count.
      clauses
       repeat. repeat :- repeat.
       count :- assert(counter(0)),fail.
       count :- repeat, counter(X), Y=X+1, retract(counter(X)),
asserta(counter(Y)), write(Y,"\n"), Y=100.
```

*Приклад 3*. Зчитування БД з файлу

Таблиця множення, сформована в прикладі 3, прочитується метою consult("tabl.txt") з файлу tabl.txt і виводиться на термінал.

```
domains
i=integer
database
prod(i,i,i)
Goal
consult("tabl.txt"), prod(X,Y,Z).
```

**Завдання для виконання**

*Завдання 1*. Сформувати базу даних хворих у вигляді сукупності термів виду хворий (прізвище, хвороба, число, кабінет). Визначити кількість хворих, що відвідали бібліотеку 10- го числа.

*Завдання 2*. Сформувати базу даних студентів у вигляді сукупності термів виду хворий (прізвище, курс, номер заліковки). Визначити кількість хворих, що вчаться на першому курсі.

*Завдання 3*. Сформувати базу даних поїздів у вигляді сукупності термів виду хворий (номер_поїзду, напрямок, число, час). Визначити кількість поїздів, що їдуть до Сімферополя.

## ЛАБОРАТОРНА РОБОТА №8

**Тема**: Операції на графах

**Мета**: розвинути вміння обробляти графи у Турбо-Пролозі.

**Пояснення** до практичної роботи: студенту необхідно знати теоретичний матеріал восьмої лекції та підготуватися до відповіді на запропоновані викладачем питання. Показати зроблені програми викладачу.

*Література*: [1], [4], [5].

### Питання для перевірки

1. Загальні поняття
2. Подання орієнтованих графів у Турбо-Пролозі
3. Побудова остовного дерева
4. Операції на графах

### Короткі теоретичні відомості

*1. Загальні поняття*

Зазвичай графом називають пару множин: безліч вершин і безліч дуг (безліч пар з безлічі вершин) Розрізняють орієнтовані та неорієнтовані графи. У орієнтованому графі кожна дуга має напрям ( розглядаються впорядковані пари вершин). Графічно зазвичай прийнято зображати вершини графа точками, а зв’язки між ними – лініями, що з’єднують вершини.

Шляхом називається послідовність вершин, з’єднаних дугами. Для орієнтованого графа напрямок шляху повинен співпадати з напрямом кожної дуги, що належить шляху. Циклом називається шлях, у якого збігаються початок і кінець.

Дві вершини орієнтованого графа, з’єднані дугою, називаються батьком і сином (або головної і підлеглої вершинами). Відомо, що якщо граф не має циклів, то обов'язково знайдеться хоча б одна вершина, яка не є нічиїм сином. Таку вершину називають кореневою. Якщо з однієї вершини досяжна інша, то перша називається предком, друга – нащадком.

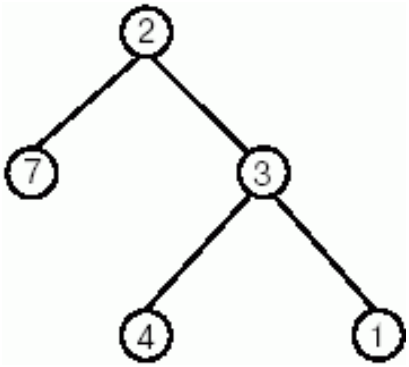
Деревом називається граф, у якого одна вершина коренева, решта вершини мають тільки одного батька і всі вершини є нащадками кореневої вершини.

Листом дерева називається його вершина, яка не має синів. Кроною дерева називається сукупність всіх листів. Заввишки дерева називається найбільша довжина шляху від кореня до листа.

Іноді зручно використовувати наступне рекурсивне визначення бінарного дерева: дерево або порожнє, або складається з кореня та лівого і правого піддерев, які в свою чергу також є деревами.

У вершинах дерева може зберігатися інформація будь-якого типу. Для простоти будемо вважати, що в вершинах дерева розташовуються цілі числа. Тоді відповідний цьому визначенню опис альтернативного домену буде виглядати наступним чином:

```
DOMAINS
tree=empty;tr(i,tree,tree)
tr(2,tr(7,empty, empty),tr(3,tr(4,empty,empty),
tr(1,empty,empty))).
```




*2. Представлення орієнтованих графів у Пролозі*

Спосіб 1.

Кожна дуга графа записується у вигляді окремої пропозиції. Наприклад

```
arca(a,b). arca(b,c). – Рис.2 (а).
```

або (граф зі зваженими дугами)

```
arca(s,t,1). arca(t,v,3). arca(v,u,2) – Рис.2(b).
```

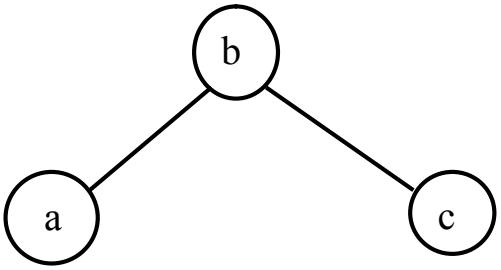



Рис. 2(а)

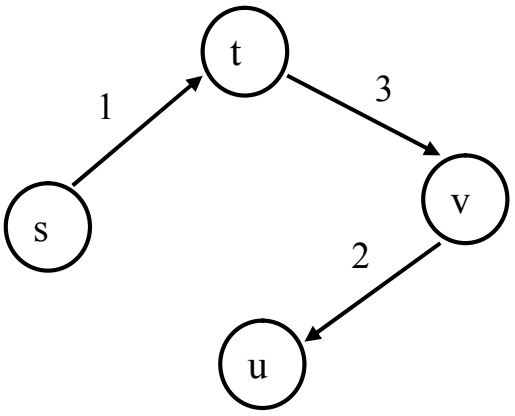



Рис. 2(b)

Спосіб 2.

Граф представляється у вигляді списку дуг. Наприклад:

G = [ arca(a,b), arca(b,c), arca(b,d), arca(c,d)] – Рис. 3(a)

або

G = [ arca(s,t,3), arca(t,v,1), ara(v,u,2), arca(u,t,5), arca(t,u,2)] – Рис. 3 (b).

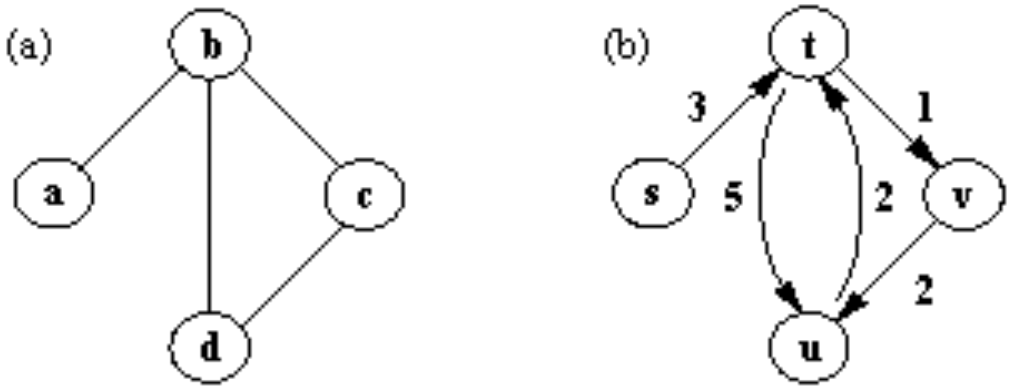



Рис. 3

Спосіб 3.

Граф представляється як один об'єкт. Графові відповідає пара множин: безліч вершин і безліч дуг. Для об'єднання множин в пару застосовується функтор graph, а для запису дуги - arca. Наприклад

G = graph([a,b,c,d], [ arca(a,b), arca(b,d), arca(b,c), arca(c,d)]) – Рис.3(a).

Усюди, де це можливо, для простоти запису програми представлятимемо графи способом 1 або способом 2.

## Приклади виконаних завдань

*Приклад 1.* Почнемо з реалізації предиката, який перевірятиме приналежність значення дереву. Предикат матиме два аргументи. Першим аргументом буде початкове значення, другий - дерево, в якому ми шукаємо дане значення.

```
domains
tree=empty;
tr(integer,tree,tree)
predicates
tree_member(integer,tree)
clauses
tree_member(X,tr(X,_,_)):-!.
tree_member(X,tr(_,L,_)):- tree_member(X,L),!.
tree_member(X,tr(_,_,R)):- tree_member(X,R).
```

## Завдання до виконання

*Завдання 1.* Описати граф. Задати відносини, що дозволяють визначити наявність в графі:

2.1. шляхів між довільною парою вершин;

2.2. багатокутників із заданим числом сторін ( наприклад, чотирикутників).

*Завдання 2.* Відношення likes ("подобається") і can_buy ("може купити").

Описати вказані відносини для наступних комбінацій "суб'єкти - предмети": суб'єкти - фрукти; суб'єкти - марки автомобілів; суб'єкти - фільми; суб'єкти - книги.

# ЛІТЕРАТУРА


1. Братко И. Алгоритмы искусственного интеллекта на языке Prolog. 3-е изд. М.: Издательский дом «Вильяме», 2001. 640 с.
2. Введение в язык логического программирования Пролог. URL: http://www.intuit.ru/department/pl/plprolog/1/.
3. Ин Ц., Соломон Д. Использование Турбо-Пролога. М.: Мир, 1993. 608 с.
4. Логические основы Пролога. URL: http://www.intuit.ru/department/pl/plprolog/2/.
5. Методичні вказівки щодо виконання лабораторних робіт з курсу «Технологія програмування та створення програмних продуктів». Частина 1. Логічне програмування / Укл. Баклан І.В., Степанкова Г.А. К.: НАУ, 2009. 44 с.
6. Скляров В.А. Программное и лингвистическое обеспечение персональных ЭВМ. Системы общего назначения: Справ. пособ. Минск: Высш. шк., 1992. 462 с.
7. Спірін О. М. Початки штучного інтелекту: Навчальний посібник для студ. фіз.-мат. спец.-тей вищих пед. навч. закл-ів. Житомир: Вид-во ЖДУ, 2004. 172 с.
8. Списки. URL: www.intuit.ru/department/pl/plprolog/7/.
9. Хабаров С.П. Интеллектуальные информационные системы. PROLOG-язик разработки интеллектуальных и экспертных систем: учебное пособие. СПб.: СПбГЛТУ, 2013. 138 с.
10. Шаров С.В., Лубко Д.В. Осадчий В.В. Інтелектуальні інформаційні системи: навч. посіб. Мелітополь: Вид-во МДПУ ім. Б. Хмельницького, 2015. 144 с
11. Шаров С.В., Лубко Д.В. Використання рекурсії при вивченні мови логічного програмування Turbo Prolog. *Наукові записки. Серія: Проблеми методики фізико-математичної і технологічної освіти*. 2015. Т.3. №8. С. 68–73.
12. Шаров С. В., Мусаєва А. Е. Огляд інструментальних засобів для створення експертної системи. *Інформаційні технології в освіті та науці збірник наук. праць*. 2016. №8. С. 199–203.
13. Шрайнер П.А. Основы программирования на языке Пролог: курс лекций. М.: Интернет - Ун-т Информ. Технологий, 2005. 176 с.
14. Чернишова Е. О. Логічне програмування з прологом. *Фізико-математичні записки: зб. наук. пр.* 2014. С. 85-88.
15. Bramer M. Logic Programming with Prolog. Springer, 2005. 228 p.

Шаров Сергій Володимирович

# МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З КУРСУ «ОСНОВИ ЛОГІЧНОГО ТА ФУНКЦІОНАЛЬНОГО ПРОГРАМУВАННЯ»

Підписано до друку: 20.12.17
Формат 60х90/16. папір офсет. Гарнітура Times New Roman.
Друк різограф. Ум. друк. арк. 1,69 Обл. вид. арк. 1,6
Тираж 100. Зам. № 195

Видавництво: РВЦ МДПУ
вул. Гетьманська, 20, м. Мелітополь, Запорізької обл., 72312
(0619)440363