Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет електроніки та комп’ютерних технологій

Методичні рекомендації щодо виконання лабораторних робіт з курсу «Експертні системи»

для студентів напряму 12 «Інформаційні технології»
факультету електроніки та комп’ютерних технологій

Уклав: В. А. Грабовський

Львів 2023

Рекомендовано до друку
кафедрою оптоелектроніки та інформаційних технологій,
протокол №2 від 8 березня 2023 р.

Методичні рекомендації щодо виконання лабораторних робіт з курсу «Експертні системи» (для студентів напряму напряму 12 «Інформаційні технології»» факультету електроніки та комп'ютерних технологій) / Укл.: В. А. Грабовський – Львів : ЛНУ імені Івана Франка, 2023 р. – 158 с.

Методичні рекомендації містять вказівки щодо виконання лабораторних робіт з курсу «Експертні системи» і призначені для студентів напряму підготовки напряму 12 «Інформаційні технології» факультету електроніки та комп'ютерних технологій. Представлені завдання щодо виконання курсу лабораторних робіт дають студентам можливість ознайомитися з особливостями створення експертних систем та використання для цього середовища програмування на основі правил CLIPS – програмного продукту, призначеного для побудови експертних систем продукційного типу. Також описані вимоги щодо представлення результатів виконання робіт, вимоги до оформлення звітів за виконану роботу та їх захисту.

Зміст

Лабораторна робота 1.

Тема: ОСВОЄННЯ ОСНОВНИХ НАВИЧОК РОБОТИ З CLIPS 6.

Мета роботи: Ознайомитися з особливостями середовища програмування CLIPS . Освоїти режими роботи в CLIPS.

Завдання до роботи:

- Ознайомитися з властивостями середовища програмування CLIPS, його архітектурою та особливостями роботи в ньому.
- Вивчити графічний інтерфейс CLIPS та операції, які можна виконувати за його допомогою.
- Освоїти режими роботи середовища – з використанням простого текстового інтерфейсу командного рядка, вбудованого та зовнішнього редактора, з використанням записаного файлу з програмою.

Вихідні матеріали:

1.1. СЕРЕДОВИЩЕ CLIPS – ІНСТРУМЕНТ ПОБУДОВИ ЕКСПЕРТНИХ СИСТЕМ ПРОДУКЦІЙНОГО ТИПУ.

CLIPS є середовищем (мовою) програмування, що дозволяє використовувати низку підходів, які забезпечують підтримку програмування на як основі правил, так і процедурного та об’єктно-орієнтованого програмування. Завдяки цьому CLIPS надає розробникам можливість застосовувати три механізми подання знань: евристичний, процедурний та об’єктно-орієнтований.

Евристичний підхід заснований на використанні правил, що визначають набір дій, які необхідно виконати в даній ситуації. Розробники експертної системи

при цьому підході задають набір правил, застосування яких дозволяє розв'язувати задачі, для вирішення яких вона призначена. Правило складається з двох частин: *антецедентної* – набору умов, які повинні бути задоволені для активування правила, і *консеквентної* – набору дій, які виконуються у разі активування правила.

CLIPS, як і більшість традиційних мов програмування (таких, наприклад, як Pascal або ж C), також підтримує і процедурний механізм представлення знань. Процедурні частини призначеного для користувача коду, що виконується CLIPS у відповідних ситуаціях, містять вбудовані або створені користувачем функції, родові функції і обробники повідомлень. Функції дозволяють створювати нові виконувані елементи, що здійснюють корисні другорядні дії або ж повертають деяке значення. Обробники повідомлень дозволяють задати поведінку об'єкта у відповідь на одержувані ним повідомлення.

Для організації об'єктно-орієнтованого підходу до подання знань у CLIPS включена власна об'єктно-орієнтована мова *COOL – CLIPS Object Oriented Language*, яка має 17 системних класів; хоч вона написана на мові C, її інтерфейс набагато ближчим до мови програмування LISP. Всі класи, визначені користувачем, є похідними від класу *User*, який певною мірою виконує функції метакласу. У ньому реалізовані практично всі базові обробники ініціалізації і видалення об'єктів. Всі класи, крім класу *Initial-Object*, є абстрактними і служать для визначення родових операцій і структур даних.

Основним підходом представлення знань в CLIPS є евристичний.

Призначена для побудови експертних систем, які ґрунтуються на правилах, мова CLIPS володіє певними фундаментальними особливостями, які притаманні саме таким системам. Основою бази знань у таких ЕС є правила, які описують властивості предметної області задачі, що розглядається; застосування цих правил механізмом логічного виведення (вирішувачем) регламентується фактами, які вводяться у систему і стосуються умов поставленої задачі. Ці особливості накладають певні умови на архітектуру CLIPS, яка містить три фундаментальні компоненти – базу фактів, базу правил і механізм їх застосування для вирішення поста-

вленої задачі.

Першим складником, необхідним для роботи експертної системи в CLIPS, є факти, які формуються з полів, які можуть бути символом, рядком символів, цілим числом або числом з плаваючою крапкою. Перше поле факту зазвичай використовується для вказівки типу інформації, що зберігається у факті, і називається *ім'ям відношення.* В CLIPS розрізняють т. з. *впорядковані* і *невпорядковані* факти. Для створення невпорядкованих фактів використовується спеціальна конструкція *deftemplate*; вона передбачає надання кожному факту імені відношення та привласнення імен його конкретним полям (ці поля називають *слотами*). Для введення масивів невпорядкованих фактів, які часто відіграють роль початкових знань про предметну область, служить конструкція *deffacts*.

Другим обов'язковим компонентом роботи системи в CLIPS є правила-продукції типу «ЯКЩО (умова), ТО (дія)», з яких формується база знань ЕС, що створюється. Кожне таке правило складається з лівої (антецедентної) і правої (консеквентної) частин. Як ліва, так і права частини правил можуть бути або простими, або складними (тобто складатися з однієї умови або з декількох і з однієї або декількох дій, відповідно). Іншими словами, правила можуть мати декілька шаблонів (для порівняння з фактами) і дій. Для створення (введення) правил у CLIPS призначений конструктив *defrule* з чітко прописаним синтаксисом їх створення.

Обов'язковою частиною системи CLIPS є машина логічного виведення. (Потрібно відмітити, що в цій мові реалізується тільки пряме логічне виведення і не забезпечується зворотне виведення.) Її робота полягає в наступному. Правила, антецедентні частини яких задовольняються введеними фактами (тобто коли перевірка умовної частини правила на відповідність наявним у пам'яті фактам закінчується успішно), відбираються і поміщаються в робочий список правил. Для забезпечення можливості погодження наявних факти з більш ніж з одним полем в умовній частині правила передбачене використання багатозначних змінних і підстановлювальних символів. Для виключення можливості постійного відбирання

одних і тих же правил під дією вже сприйнятих фактів в системі CLIPS використовується механізм релаксації.

У складних експертних системах, бази знань яких містять велику кількість правил, можлива ситуація, коли за наявними в робочій пам'яті фактами вирішувачем може бути відібрано і поміщено в робочу пам'ять ЕС більш ніж одне правило – тоді виникає певний «конфлікт інтересів» між ними щодо того, яке ж із відібраних правил має бути виконане першим. Для вирішення подібних ситуацій у CLIPS передбачено сім вбудованих стратегій вирішення таких конфліктів.

Загалом, CLIPS як інструментальний засіб для побудови експертних систем є потужним інструментом, який дозволяє з мінімальними затратами розробити потрібну експертну систему. До невід'ємних переваг середовища можна віднести його кросплатформність, що дозволяє встановити розроблену експертну систему практично на будь-який комп'ютер, а також безкоштовне розповсюдження та зручний для роботи інтерфейс.

1.1.1. Основні поняття та положення CLIPS.

Прототип мови CLIPS був розроблений в 1985 р. в космічному центрі NASA (США).

Спочатку абревіатура розшифровувалася як «*C Language Integrated Production System*» (мова С, інтегрована з продукційними системами). З часом мова була неодноразово модернізована – в неї введені процедурні і об'єктно-орієнтовані парадигми (починаючи з версії 5), підтримка модульної структури програм, тощо. Сьогодні CLIPS являє собою призначений для створення продукційних експертних систем сучасний програмний інструмент, що складається з інтерактивного середовища – експертної оболонки зі своїм способом подання знань, гнучкої й потужної мови програмування та ще декількох допоміжних засобів – і поширюється безкоштовно.

При роботі ми використовуватимемо головно версію CLIPS 6.30, яка доступна на сайті *https://sourceforge.net/projects/clipsrules/files/CLIPS/6.30/,* або CLIPS

6.31, в яку внесені деякі вдосконалення і яку можна завантажити на *https://sourceforge.net/projects/clipsrules/files/CLIPS/6.31/*; ні загальний вигляд інтерфейсу, ні його функціонал у цих версіях фактично не відрізняються.

Встановити CLIPS на комп'ютер можна двома способами: для версії 6.30 потрібно завантажити і запустити виконавчий файл (для Windows це,в залежності від розрядності ОС, clips_windows_64_bit_executables_630.msi чи clips_windows_32_bit_executa-bles_630.msi, а для 6.31 – завантажити CLIPS-IDE631LF та запустити CLIPSIDE32LF.exe; інший метод – інсталювати CLIPS, використавши відповідні інсталяційні файли.

В ОС WINDOWS середовище CLIPS завантажується запуском файлу CLIPSIDE64.exe (версія 6.30) або CLIPSIDE32LF.exe (версія 6.31).

Як уже згадувалося вище, CLIPS підтримує три основні способи представлення знань:

- продукційні правила для представлення евристичних знань;
- функції для представлення процедурних знань;
- об'єктно-орієнтоване програмування.

Вигляд вікна оболонки CLIPS (версії 6.31 6/12/19) показаний на рис. 1.1.

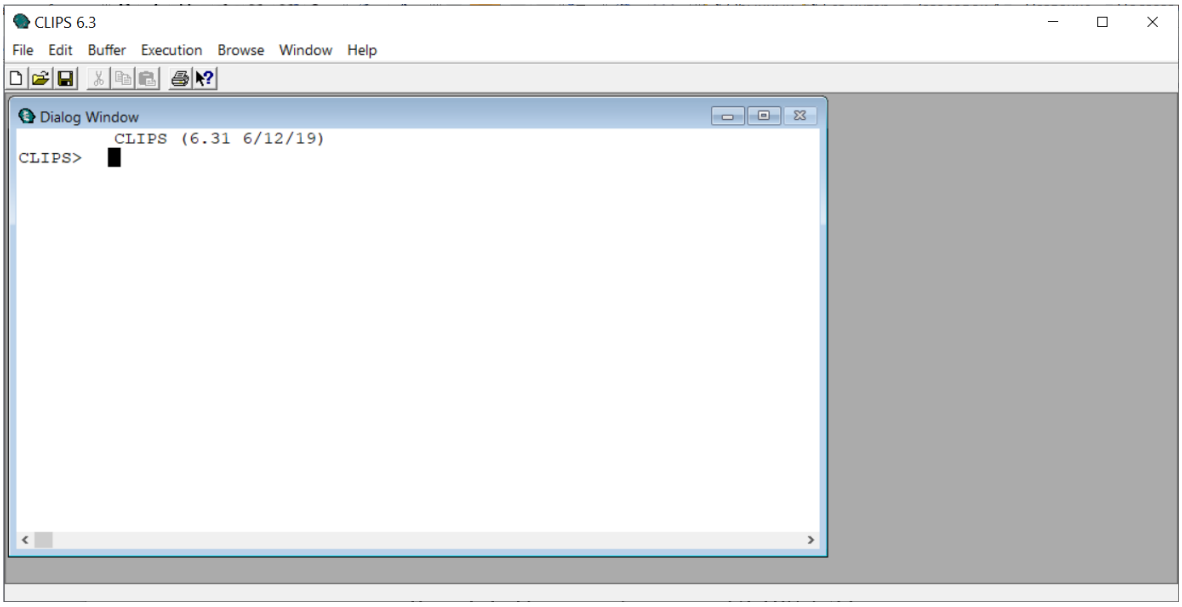

Рис. 1.1. Вигляд оболонки CLIPS після запуску (версія 6.31).

Після запуску на моніторі з'являється діалогове вікно зі стандартним рядком запрошення CLIPS>, в якому і вводяться команди. Потрібно пам'ятати, що CLIPS не інтерпретує кириличний шрифт, тому всі факти, конструкції і правила потрібно вводити, використовуючи латинський шрифт.

Призначення основних пунктів меню віконного інтерфейсу, які найчастіше використовуються при роботі в середовищі CLIPS, представлені в табл. 1.1.

Таблиця 1.1. Основні команди головного меню CLIPS

Пункт	Підпункт	"Гарячі" клавіші	Призначення команди
File	New Open Load … Load Batch	Ctrl+N Ctrl+O Ctrl+L	Створення нового файлу Відкриття файлу Завантаження конструкцій з файлу. Виконання пакетного файлу
Edit	Cut Copy Paste	Ctrl+X Ctrl+C Ctrl+V	Вирізання фрагмента Копіювання фрагмента Вставка рядка з буфера обміну
Execution	Reset Run Step	Ctrl+U Ctrl+R Ctrl+T	Ініціалізація конструкцій Запуск на виконання Виконання одного кроку виведення
Browse	Module Defrule Manager Deffacts Manager		Відображає модуль Менеджер правил Менеджер фактів
Window	Facts Window Agenda Window Clear dialog window		Активування вікна списку фактів Активування вікна агенди Очищає вікно з командним рядком

1.1.2. Режими роботи CLIPS.

Експертні системи, створені за допомогою CLIPS, можуть бути запущені трьома основними способами:

- уведенням відповідних команд і конструкторів мови безпосередньо в середовище CLIPS;
- з використанням інтерактивного віконного GUI-інтерфейсу CLIPS (для версій в ОС Windows або Macintosh);
- за допомогою програм-оболонок, що реалізують свій інтерфейс спілкування з користувачем і використовують механізми створення баз знань і логічного виведення CLIPS.

При введенні команд з командного рядка важливо пам’ятати, що:

- кожна команда CLIPS повинна мати погоджену кількість відкриваючих и закриваючих круглих дужок;
- будь-які символи, не поміщені в дужки, інтерпретатором CLIPS сприймаються як константи.

При виклику функції в CLIPS потрібно враховувати префіксну форму їх представлення – аргументи функції можуть стояти тільки після її назви. Виклик функції починається з дужки, що відкривається, за якою слідує ім’я функції; потім слідують аргументи, кожен з яких відокремлений одним або декількома пробілами. Аргументами функції можуть бути дані простих типів, змінні або ж виклики інших функцій. В кінці виклику ставиться дужка, що закривається. Наприклад, вираз 3+8*12+7 в CLIPS записується таким чином:

(+ 3 (* 8 12) 7).

1.1.3. Особливості роботи в CLIPS.

Вигляд діалогового вікна CLIPS («Dialog Window»), яке використовується для введення і виконання команд, а також виведення отриманого результату в командній стрічці, представлений на рис 1.1. Це свого роду командний рядок обо-

лонки, де можна виконати будь-яку команду CLIPS, записавши її код після запрошення CLIPS> і натиснувши клавішу <Enter>.

Наприклад, написавши в командному рядку після запрошення наступний код (включаючи дужки):

(printout t “Hello CLIPS” crlf)

і натиснувши <Enter>, ми змусимо CLIPS виконати функцію виведення на екран і отримаємо рядок «Hello CLIPS», відразу після нашого виклику; після чого система повернеться в режим очікування.

Потрібно мати на увазі синтаксис команди *printout*, який має вигляд:

(printout <logical-name><print-items>*),

де параметр <logical-name> показує пристрій призначення виведення команди *printout* (тобто пристрій, на який виводиться результат), а параметр <print-items>* містить елементи (від нуля і більше), які повинні бути виведені за допомогою вказаної команди.

Як уже згадувалося, у WINDOWS-версії CLIPS передбачена можливість написання коду програми (або ж її елементів) з використанням вбудованого редактора. Для того, щоб увійти в нього і почати писати код програми, потрібно створити новий файл шляхом вибору пункту New з меню File (або одночасно натиснувши комбінацію клавіш Ctrl+N; на екрані з’явиться вікно введення (вікно Untitled1 на рис. 1.2), в якому і потрібно набирати необхідний код. Після набору коду команди, будь-якої конструкції CLIPS чи коду всієї програми її можна запустити на виконання; для цього потрібно попередньо «замаркувати» написаний код і натиснути кнопку Batch Selection з меню Buffer (або ж одночасно натиснувши комбінацію клавіш Ctrl+M).

Виконання програми на CLIPS принципово відрізняється від виконання програми, написаній на класичній мові програмування (такій як, наприклад, C або Pascal). На відміну від вказаних програм, де після введення команди *run* відбувається компіляція програми з її наступним виконанням, запуск програми на CLIPS фактично означає запуск механізму логічного виведення, заснованого на викорис-

танні правил, що є в системі; вибір цих правил, своєю чергою, заснований на співставленні умовних (антецедентних) частин внесених в базу знань правил з наявними на текучий момент в оперативній пам'яті CLIPS фактами. Це зумовлює і відміну у вигляді самої програми – у загальному випадку проста програма на CLIPS складається з набору правил і функцій, основним призначенням яких є здійснення різноманітних маніпуляцій з фактами – введенням їх у пам'ять системи або ж їх виведенням з неї, видозміна старих фактів та отримання нових, тощо.

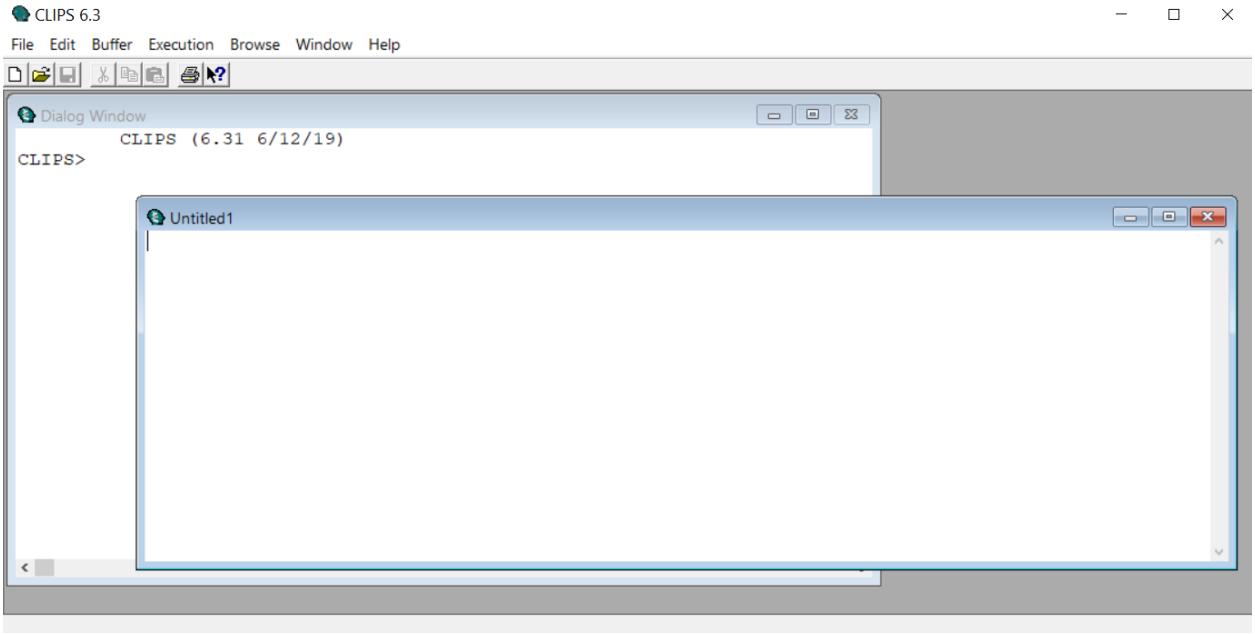

Рис. 1.2. Вікно вбудованого редактора CLIPS.

1.2. Інтерфейс CLIPS (Версія 6.30/31 для ОС Windows).

Як уже згадувалося, написання коду, введення будь-якої команди, програми чи її частини та їх виконання в CLIPS можна здійснити кількома способами. Команди можна ввести безпосередньо у діалогове вікно середовища у спосіб, подібний до стандартного інтерфейсу командного рядка; інший спосіб – введення через вбудований редактор, який реалізований у спосіб, подібний до інших редакторів

Windows. Особливості виконання деяких команд будуть розглянуті нижче.

Однак, у версіях CLIPS, працюючих в ОС Windows та MacOC, передбачене дублювання введення потрібних команд за допомогою графічного інтерфейсу. При користуванні графічним інтерфейсом потрібно пам'ятати, що деякі команди його меню доступні, поки CLIPS виконує користувацьку програму. Серед цих команд є пункт меню «Options», пункт меню «Watch», пункт меню «Turn Dribble On/Turn Dribble Off» та всі елементи меню в меню «Window». Крім того, вікна можуть бути переміщені та змінені за розміром під час виконання CLIPS. Докладнішу інформацію щодо графічного інтерфейсу CLIPS та доступності операцій з його меню можна отримати в [1].

Розглянемо окремі пункти меню інтерфейсу CLIPS (версія 6.30 для ОС Windows) докладніше.

1.2.1 Меню File.

Після натискання на кнопку *File* у вікні CLIPS з'явиться випадне вікно цього меню (рис. 1.3). Дія кожної з команд така:

- Команда *New* (Ctrl+N) відкриває новий буфер вбудованого редактора з ім'ям вікна Untitled.
- Команда *Open* (Ctrl+O) дозволяє користувачеві вибрати текстовий файл, який буде відкритий у вбудованому редакторі для редагування.
- Команда *Load* (Ctrl+L) (еквівалентна команді CLIPS (load <file-name>) дозволяє користувачеві вибрати текстовий файл, який буде завантажений у базу знань. Коли ця команда буде обрана та вибрано файл, відповідна команда завантаження CLIPS буде перезаписана до діалогового вікна та виконається. Ця команда недоступна, коли CLIPS виконує програму.
- Команда *Load Batch* дозволяє користувачеві вибрати текстовий файл, який буде виконуватися як пакетний файл. Ця команда еквівалентна команді CLIPS (batch <файл-ім'я>). Коли ця команда буде обрана та вибрано файл,

відповідна команда CLIPS буде повторена в діалоговому вікні та виконана. Ця команда недоступна, коли CLIPS виконує програму.

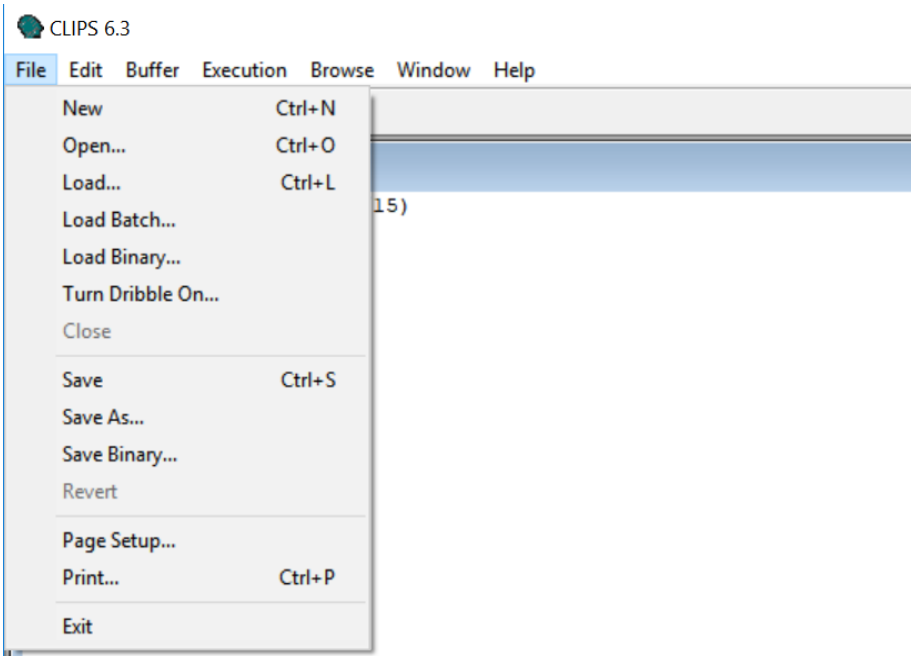

Рис. 1.3. Вигляд випадного вікна меню *File*.

- Команда *Load Binary* дозволяє користувачеві вибрати файл, який буде завантажений як бінарне зображення. Ця команда еквівалентна команді CLIPS (bload <file-name>). Коли ця команда вибирається і файл виділений, відповідна команда CLIPS bload буде повторюватися в діалоговому вікні та виконуватися. Ця команда недоступна, коли CLIPS виконує програму.
- Команда *Turn Dribble On* дозволяє користувачеві вибрати текстовий файл, в якому буде зберігатися наступне вікно дисплея. Ця команда еквівалентна команді CLIPS (dribble-on <file-name>). Коли команда буде обрана і вибрано файл, відповідна команда CLIPS dribble-on буде повторюватися в діалоговому вікні та виконуватися. Поки файл dribble активний, цей пункт меню буде змінено для читання.
- Команда *Close* закриває поточно активне вікно.
- Команда *Save* (Ctrl+S) зберігає файл, який є у активному вікні редактора або у

діалоговому вікні, як пакетний або текстовий файл. Пакетні документи CLIPS виконуються як серія команд при запуску, тоді як документи текстових файлів CLIPS розміщуються в буфері редагування при запуску. Обидва типи документів можна редагувати як текстові файли у редакторі.

- Команда *Save As* дозволяє активне вікно редагування зберігати під новим ім'ям.
- Команда *Save Binary* (еквівалентна команді CLIPS (bsave <file-name>) дозволяє конфігураціям, які в даний час зберігаються в CLIPS, бути збережені як бінарний файл. Ця команда недоступна, коли CLIPS виконує програму.
- Команда *Revert* відновлює активне вікно редагування до останньої збереженої версії файлу в буфері. Будь-які зміни, зроблені після останнього збереження файлу, будуть відхилені.
- Команда *Page Setup* дозволяє користувачеві вказати інформацію про розмір паперу, який використовується принтером.
- Команда *Print* (Ctrl+P) дозволяє користувачеві друкувати активне вікно редагування.
- Команда *Exit* – вихід з CLIPS. Ця команда недоступна, коли CLIPS виконує програму.

1.2.2 Меню Edit.

Після натискання на кнопку *Edit* у вікні CLIPS з'явиться випадне вікно цього меню (рис. 1.4). Дія кожної з команд така:

- Команда *Undo* (Ctrl + Z) дозволяє скасувати останню операцію редагування.
- Команда *Cut* (Ctrl + X) видаляє вибраний у вікні редагування текст та розміщує його в буфер обміну.
- Команда *Copy* (Ctrl + C) копіює виділений текст у вікні редагування і поміщає його в буфер обміну.
- Команда *Paste* (Ctrl + V) копіює вміст буфера обміну до точки вибору в вікні

редагування або в діалоговому вікні.

- Команда *Delete* видаляє вибраний у вікні редагування текст.

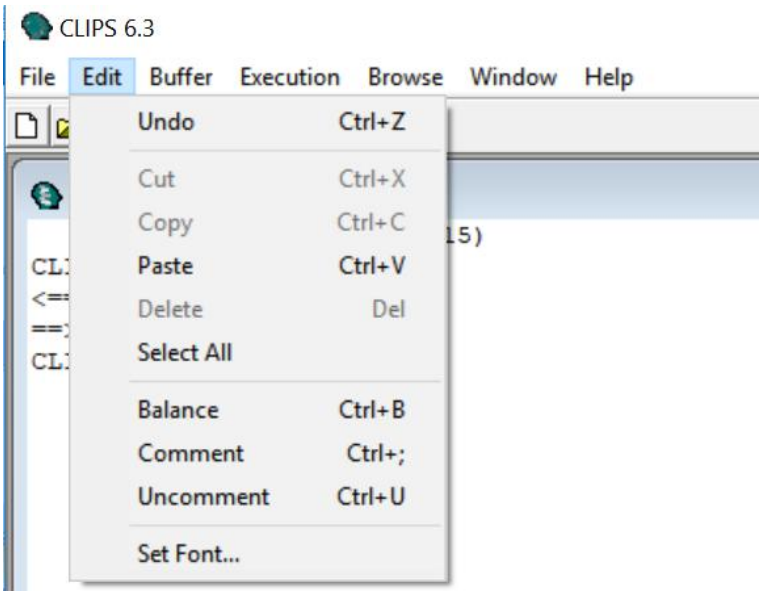

Рис. 1.4. Вигляд випадного вікна меню *Edit*

- Команда *Select all* виділяє весь текст у активному вікні редагування.
- Команда *Balance* (Ctrl + B) діє на поточному виборі в вікні редагування, намагаючись знайти найменший вибір, що містить поточний вибір, який має збалансовані дужки. Є чисто текстовою операцією.
- Команда *Comment* (Ctrl +;) діє на поточний вибір у активному вікні редагування, додавши крапку з комою до початку кожного рядка, що міститься у виділенні.
- Команда *Uncomment* (Ctrl + U) використовує поточний вибір у активному вікні редагування, видаливши крапку з комою (якщо така існує) з початку кожного рядка, що міститься у виділенні.
- Команда *Set font* дозволяє змінити шрифт, який використовується в поточному активному буфері для редагування; з’являється діалогове вікно (рис. 1.5), в якому вибираються шрифт і його потрібні параметри.

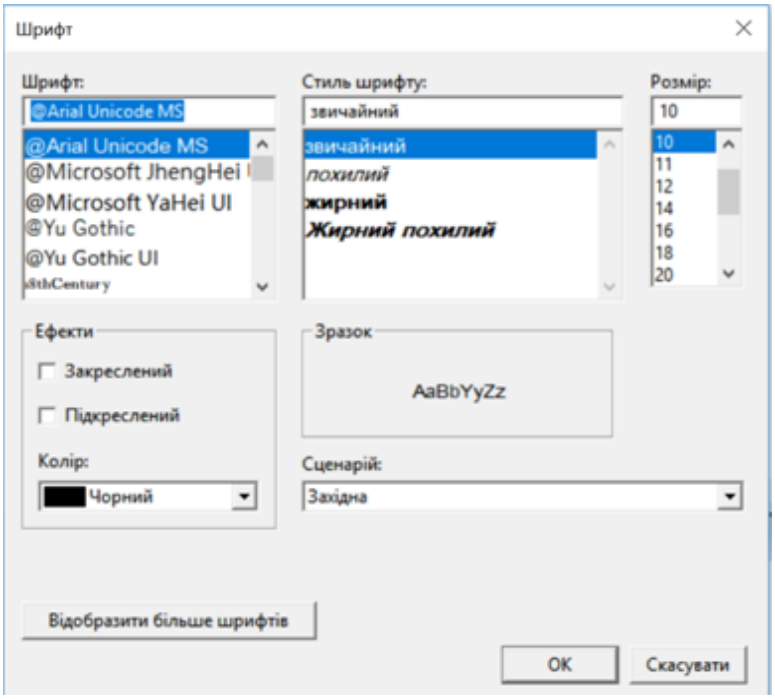

Рис. 1.5. Вигляд випадного вікна команди *Set font* меню *Edit*.

1.2.3. Меню Buffer.

Після натискання на кнопку *Buffer* у вікні CLIPS з’явиться випадне вікно цього меню (рис. 1.6). Дія кожної з команд така:

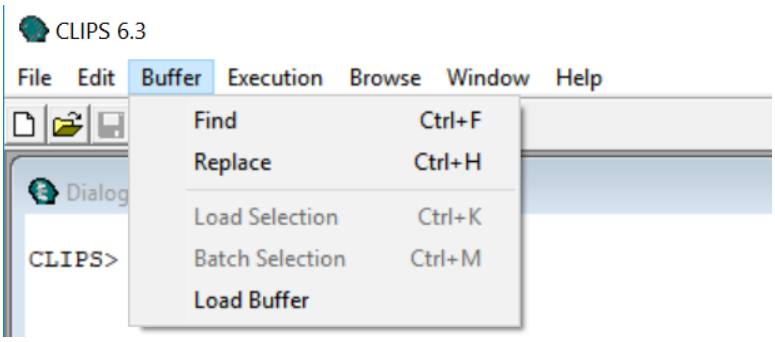

Рис. 1.6. Вигляд випадного вікна меню *Buffer*.

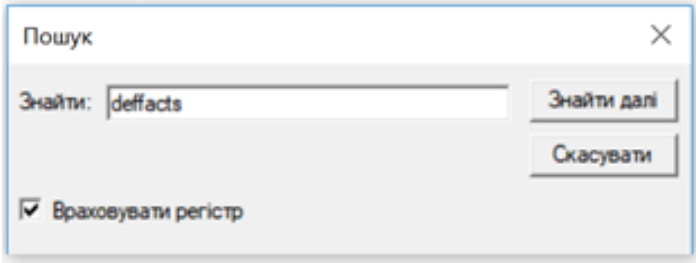

Рис. 1.7. Вигляд випадного вікна команди *Find* меню *Buffer*.

- Команда *Find* (Ctrl+F) відображає діалогове вікно (рис. 1.7), яке дозволяє користувачеві встановлювати параметри для операцій текстового пошуку. Діалогове вікно дозволяє вказувати рядок пошуку. Параметр Match Case робить операцію пошуку по рядках без чутливості для алфавітних символів; за замовчуванням цей параметр вимкнено.
- Команда *Replace* (Ctrl+H) дозволяє користувачеві встановлювати параметри для текстового пошуку та заміни операцій. Відображене діалогове вікно (рис. 1.8) дозволяє вказати рядок пошуку та заміни.

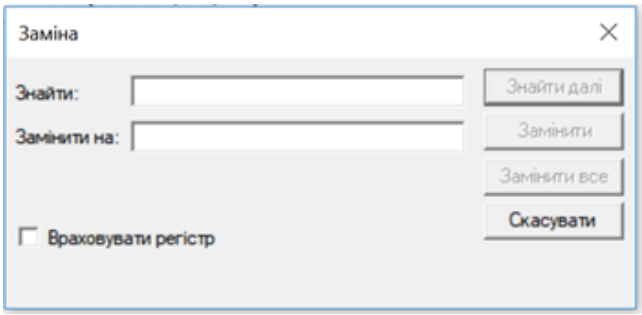

Рис. 1.8. Вигляд випадного вікна команди *Replace* меню *Buffer*.

- Команда *Load Selection* (Ctrl+K) завантажує поточний вибір з вікна редагування в базу знань CLIPS. ; команда недоступна, коли CLIPS виконує команду.
- Команда *Batch Selection* (Ctrl+M)розглядає поточний вибір у вікні редагування так, якби він був пакетним файлом, і виконує його як серію команд; команда недоступна, коли CLIPS виконує якусь програму.

- Команда *Load Buffer* завантажує вміст активного вікна редагування в базу знань CLIPS. Ця команда недоступна, коли CLIPS виконує програму.

1.2.4 Меню Execution.

Після натискання на кнопку *Execution* у вікні CLIPS з'явиться випадне вікно цього меню (рис. 1.9). Дія кожної з команд така:

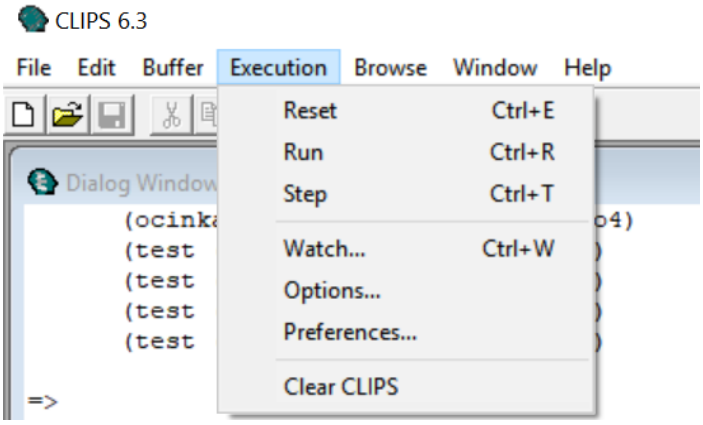

Рис. 1.9. Вигляд випадного вікна меню *Execution.*

- Команда *Reset* (Ctrl+E) еквівалентна команді CLIPS (reset). Коли ця команда буде обрана, команда CLIPS (reset) буде відтворена в діалоговому вікні та виконана. Якщо увімкнено попередження і в CLIPS є активована певна програма, з'явиться діалогове вікно з попередженням, яке дає можливість скасувати цю команду. Ця команда недоступна, коли CLIPS виконує якусь дію.
- Команда *Run* (Ctrl+R) еквівалентна команді CLIPS (run). Коли ця команда буде обрана, команда CLIPS (run) буде повторюватися в діалоговому вікні та виконуватися.
- Команда Step (Ctrl+T) еквівалентна команді CLIPS (run <limit>), де < limit> – це значення для Step Rule Firing Increment (він встановлюється за допомогою команди Options в меню Edit).

- Команда *Watch* (Ctrl+W) показує діалогове вікно підменю Watch Options (рис. 1.10), яке дозволяє користувачеві встановлювати різні елементи перегляду як включені або вимкнені, поставивши або знявши відповідний прапорець. Натискання кнопки All прапорець на всі елементи підменю, а кнопки None – скидає усі прапорці. Натискання кнопки OK виводить з діалогу та змінює поточні налаштування годинника на ті, що були виставлені користувачем. Натискання кнопки Cancel приводить до виходу з діалогового вікна, але зберігає оригінальні налаштування елементів перегляду до введення.

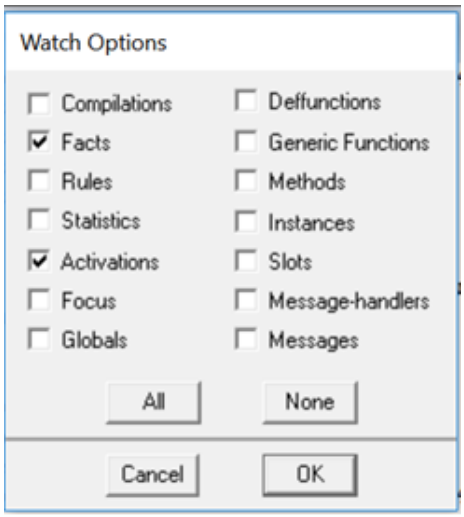

Рис. 1.10. Вигляд випадного вікна команди *Watch* меню *Execution.*

- Команда *Options* відображає діалогове вікно (рис. 1.11), яке дозволяє користувачеві встановлювати різні параметри виконання команд CLIPS, встановлюючи або знімаючи відповідний прапорець у віконці відповідної опції. Випадне меню Salience Evaluation дозволяє встановити порядок визначення поточного пріоритету правила (коли визначене, коли активоване, або в кожному циклі), а Strategy – дозволяє встановити поточну стратегію вирішення конфліктів (depth, breadth, lex, mea, complexity, simplicity чи random). Натискання кнопки OK виводить з діалогового вікна та змінює параметри виконання CLIPS для тих, що встановлюються користувачем. Натискання кнопки «Скасувати» ви-

водить із діалогового вікна зі збереженням оригінальних параметрів виконання CLIPS до ведення діалогу.

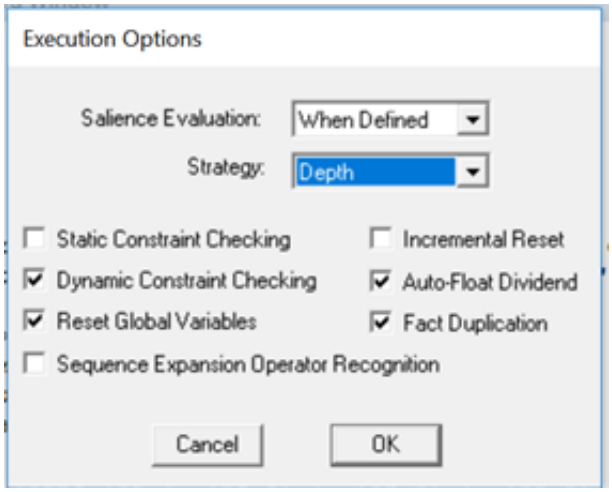

Рис. 1.11. Вигляд випадного вікна команди *Options* меню *Execution.*

Встановленням відповідних прапорців можна виставити перевірку типу значень слотів при створенні шаблонних фактів та екземплярів об’єктів. Підтримуються два типи перевірки обмежень: статична (static-constraint-checking) та динамічна (dynamic-constraint-checking). Коли увімкнено перевірку статичного обмеження, перевірки порушень обмежень перевіряються, коли обробляються виклики функції та конструкції, а коли включена перевірка динамічного обмеження, нові об’єкти даних перевіряють значення своїх слотів на порушення обмежень. Загалом, перевірка статичних обмежень відбувається, коли завантажується програма CLIPS, динамічних – під час запуску програми CLIPS. За замовчуванням увімкнено перевірку статичних обмежень, і перевірка динамічних обмежень вимкнена.

- Команда *Preferences* відображає діалогове вікно (рис. 1.12), яке дозволяє користувачеві встановлювати параметри для декількох параметрів у інтерфейсі Windows CLIPS.

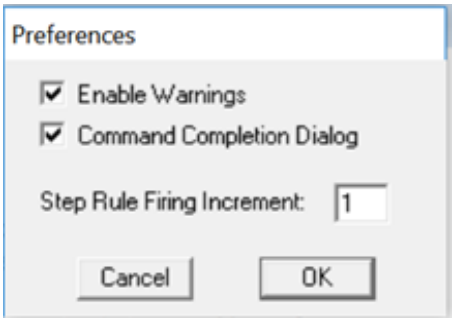

Рис. 1.12. Вигляд випадного вікна команди *Preferences* меню *Execution.*

Параметр Enable Warnings визначає, чи видаються певні попередження (за замовчуванням – увімкнено). Коли увімкнено застереження, певні команди, вибрані з рядка меню, відобразяться у спливаючому діалоговому вікні, що дає користувачеві можливість скасувати команду.

Діалогове вікно Command Completion Dialog визначає, чи відображатиметься діалогове вікно для визначення, яке з доступних більше від одного завершень використовувати (за замовчуванням – увімкнено). Тобто, діалогове вікно буде відображатися кожного разу, коли існує декілька можливих завершень. Якщо цей параметр вимкнено, звучить сигнал, що не існує можливого завершення.

Параметр Step Rule Firing Increment визначає, скільки правил буде виконуватися, коли буде видано команду Step з Меню Execution. Для цього параметра можна ввести будь-яке ціле число від 1 до 999 включно. Значення за замовчуванням для цієї опції становить 1. Ця команда доступна, коли CLIPS виконує програму.

- Команда *Clea*r CLIPS еквівалентна команді CLIPS (очистити). Коли ця команда буде обрана, команда CLIPS (Clear) буде повторюватися у діалоговому вікні та виконуватись. Якщо увімкнено Enable Warnings, у діалоговому вікні з’явиться попередження, що дасть змогу скасувати цю команду. команда недоступна, коли CLIPS виконує програму.

1.2.5. Меню Browse.

Після натискання на кнопку *Browse* у вікні CLIPS з'явиться випадне вікно цього меню (рис. 1.13). Натискання на кнопку Module зумовлює виникнення підменю *Module* (рис. 1.14), у якому відображається список модулів, що визначені в середовищі CLIPS; поточний модуль відмічений прапорцем. Вибір модуля зі списку меню змінює поточний модуль на вибраний елемент.

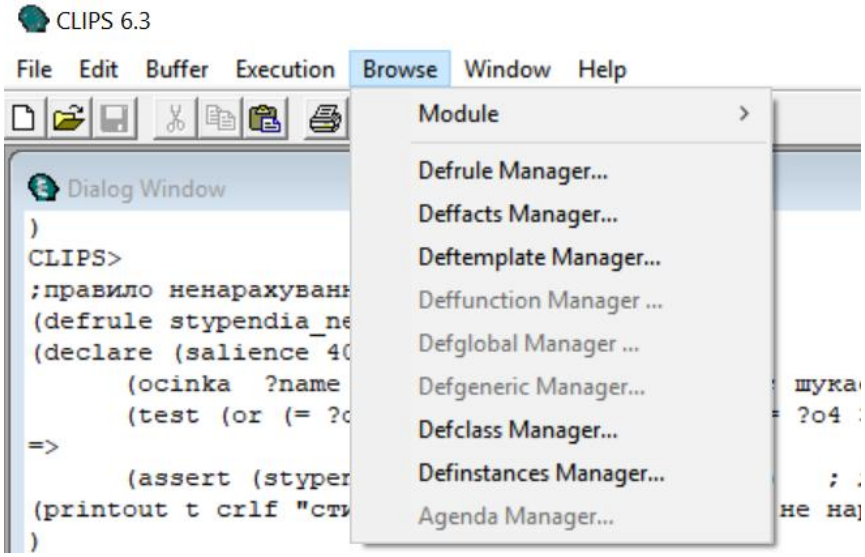

Рис. 1.13. Вигляд випадного вікна меню *Browse*.

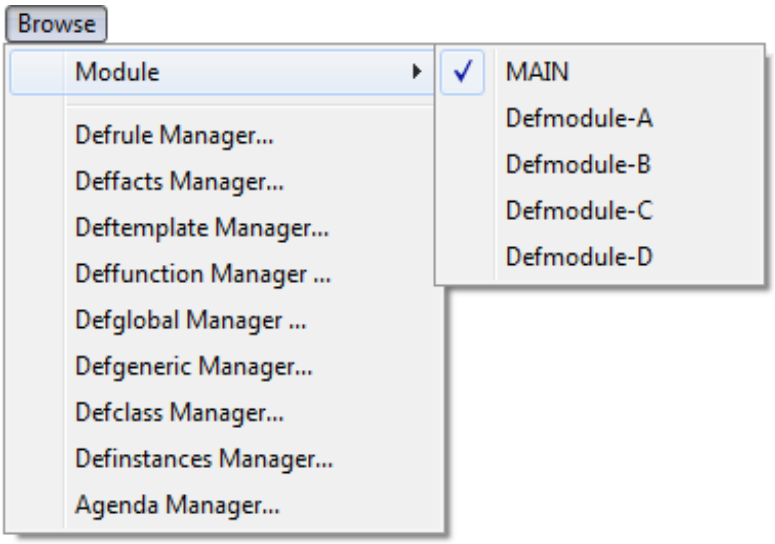

Рис. 1.14 Вигляд випадного вікна *Module* меню *Browse*.

- Команда *Defrule Manager* дозволяє переглянути наявні у базі знань правила. Діалогове вікно (рис. 1.15), що з'являється у відповідь на цю команду, дозволяє виконати з вибраною зі списку у вікні правилом операцію Remove, Refresh, Matches, чи Pprint.

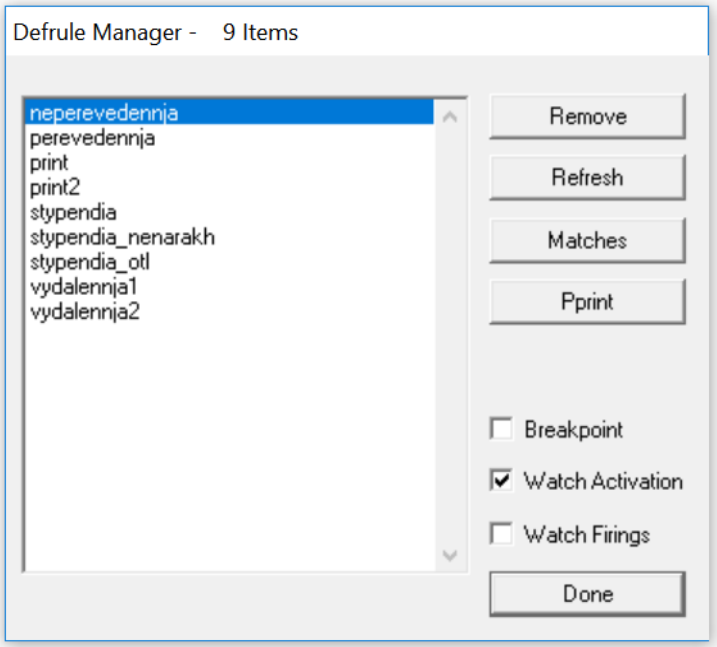

Рис. 1.15. Вигляд випадного вікна *Defrule Manager* меню *Browse*.

Прапорець Breakpoint використовується для увімкнення чи вимкнення точки зупинки на поточному виділеному режимі defrule. Правила з набором точок зупинки мають перевірити їх прапорець. Прапорець Watch Activations використовується для ввімкнення або вимкнення повідомлень про активування чи деактивування правила. Прапорець Watch Firings використовується для ввімкнення чи вимкнення повідомлень, які друкуються при виконанні правила. Щоб вийти з діалогового вікно, потрібно натиснути кнопку Done. команда Defrule Manager недоступна, коли CLIPS виконує програму.

- Команда *Deffacts Manager* дозволяє перевіряти наявні в базі знань факти, що підлягають розгляду. Діалогове вікно, яке з'являється у відповідь на цю ко-

манду (рис. 1.16), дозволяє виконувати операцію з фактами, вибраними зі списку, що знаходиться в базі знань. Натискання кнопки Remove видалить вибраний deffacts, Pprint роздрукує вибрані факти. Завершивши, натисніть кнопку Done, щоб закрити діалогове вікно. команда Deffacts Manager недоступна, коли CLIPS виконує програму.

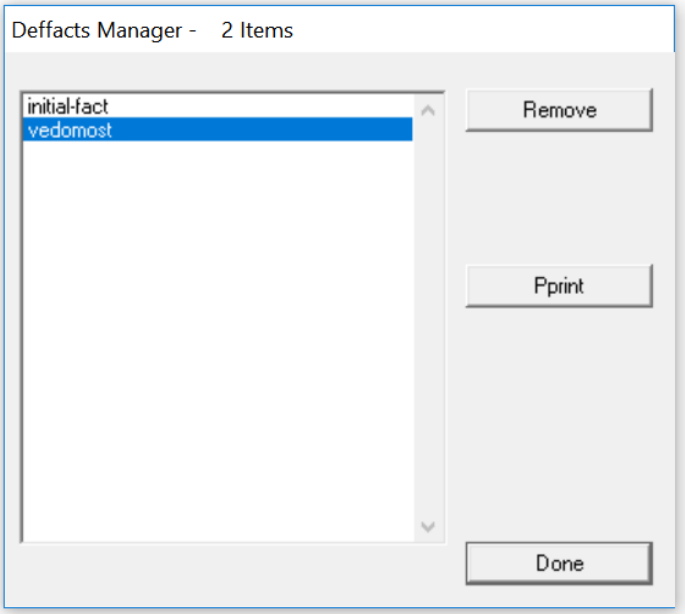

Рис. 1.16. Вигляд випадного вікна *Deffacts Manager* меню *Browse.*

- Команда *Deftemplate Manager* дозволяє переглядати наявні в базі знань шаблони deftemplate. Діалогове вікно, яке з'являється у відповідь на цю команду (рис. 1.17), дозволяє здійснити операцію видалення або друку в діалоговому вікні обраного зі списку deftemplate. Прапорець Watch використовується для включення або відключення повідомлення, які друкуються кожен раз, коли факт вводиться або видаляється. Щоб закрити діалогове вікно, потрібно натиснути кнопку Done. команда Deftemplate Manager недоступна, коли CLIPS виконує програму.

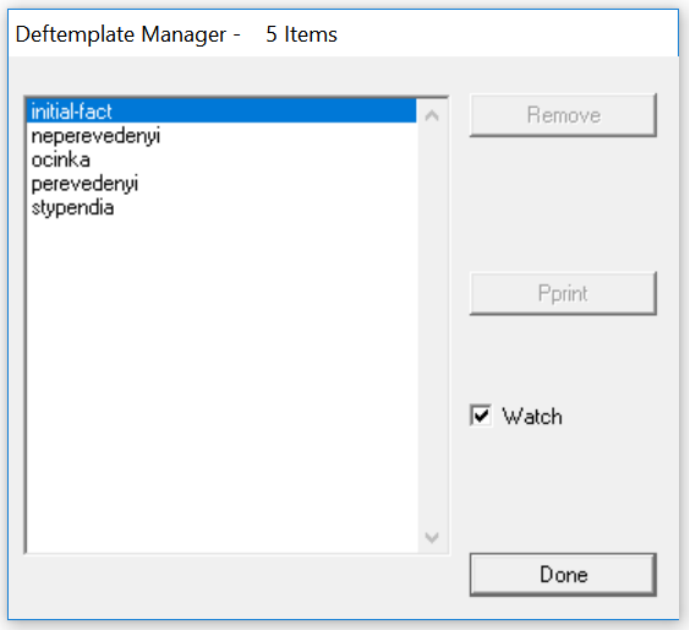

Рис. 1.17. Вигляд випадного вікна *Deftemplate Manager* меню *Browse.*

- Команда *Deffunction Manager* дозволяє переглядати deffunctions в базі знань. Діалогове вікно, яке з'являється у відповідь на цю команду (рис. 1.18) дозволяє здійснювати операцію, яка буде виконуватися над обраною зі списку deffunctions в базі знань функцією. Операції, які виконуються, аналогічні до розглянутих для Deftemplate Manager

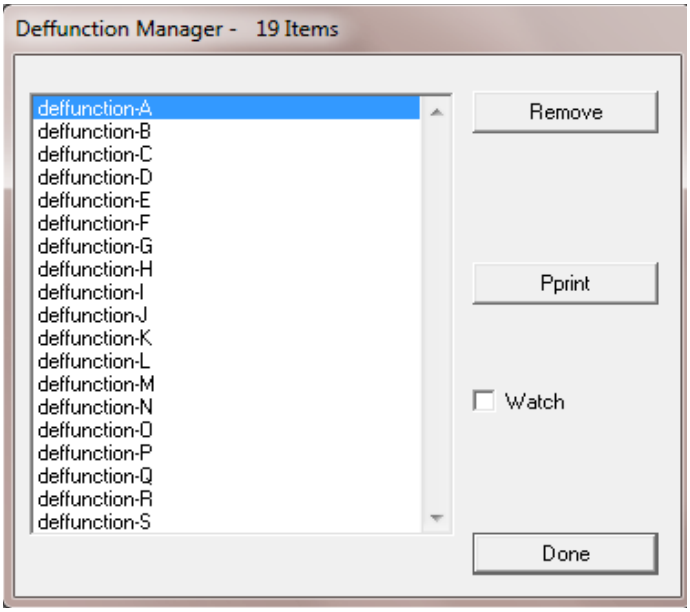

Рис. 1.18. Вигляд випадного вікна *Deffunction Manager* меню *Browse.*

- Команда *Defglobals Manager* дозволяє перевіряти глобальні змінні у базі знань. Діалогове вікно, що з'являється у відповідь на цю команду (рис. 1.19), дозволяє виконати операцію над defglobal, вибраному зі списку defglobals, які знаходяться в базі знань. Дія відповідних операцій подібні до аналогічних, розглянутих для Deftemplate Manager.

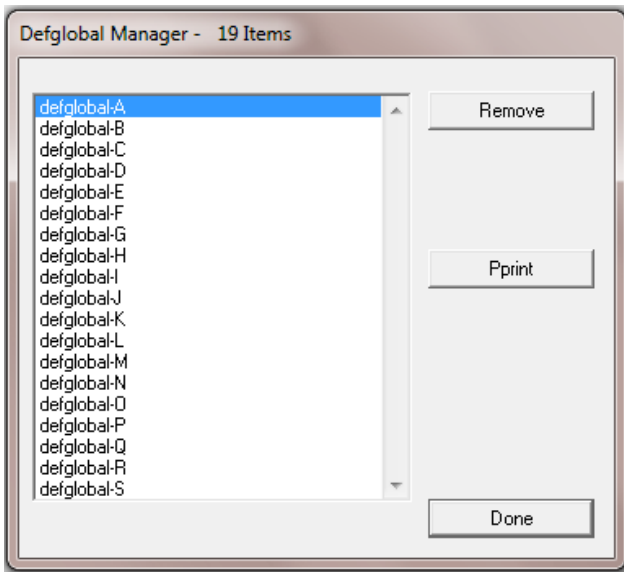

Рис. 1.19. Вигляд випадного вікна *Defglobals Manager* меню *Browse.*

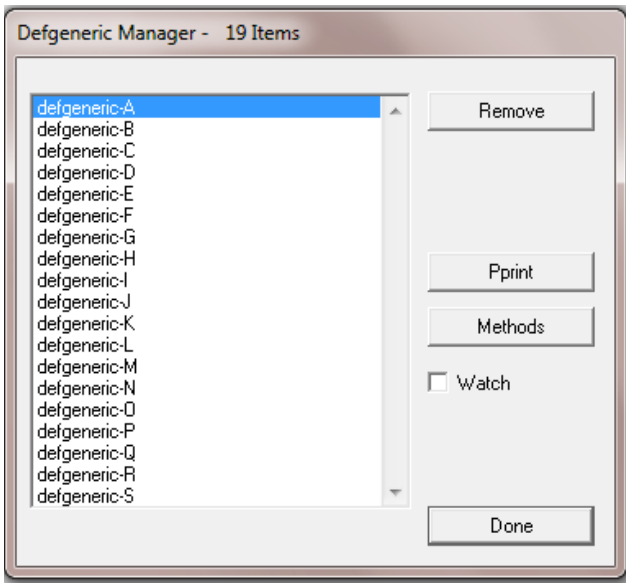

Рис. 1.20. Вигляд випадного вікна *Defgeneric Manager* меню *Browse.*

- Команда *Defgeneric Manager* дозволяє перевіряти наявні у базі знань defgenerics. Діалогове вікно (рис. 1.20), що з'являється у відповідь на цю команду, дозволяє виконати операцію з defgeneric, вибраною зі списку defgenerics, що знаходяться в базі знань. Дія операцій Remote, Pprint тут аналогічна розглянутим вище. При натисканні кнопки Methods відкривається діалогове вікно Defmethods-Handler Manager (рис. 1.21), яке дозволяє переглядати наявні в пам'яті defmethods для обраного defgeneric і виконувати зазначені у вікні операції над defmethod, вибраним зі списку. Натискання кнопки Done повертає керування у діалогове вікно Defgeneric Manager.

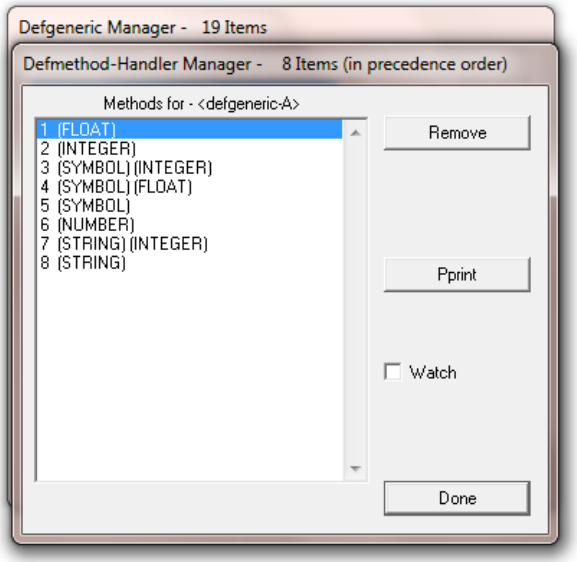

Рис. 1.21. Вигляд випадного вікна *Defmethods-Handler Manager*.

- Команда *Defclass Manager* показує діалогове вікно (рис. 1.22), яке дозволяє виконувати операцію над класами, вибраному зі списку defclasses, наявних в даний час в базі знань. Кнопки Remove, Describe та Pprint відтворюють еквівалентну команду CLIPS та вихідний результат у діалоговому вікні. Browse – відображає ієрархію класів для вибраного класу defclass. Прапорець Watch Instances використовується для увімкнення або вимкнення повідомлень, які надсилаються при створенні або видаленні екземпляра класу, а Watch Slots – для увімкнення чи вимкнення повідомлень, що друкуються, коли змінюються значення слотів у екземплярі вибраного класу. Якщо позначено прапорець для

defclass, то повідомлення будуть надруковані, коли екземпляр цього класу змінить значення слота.

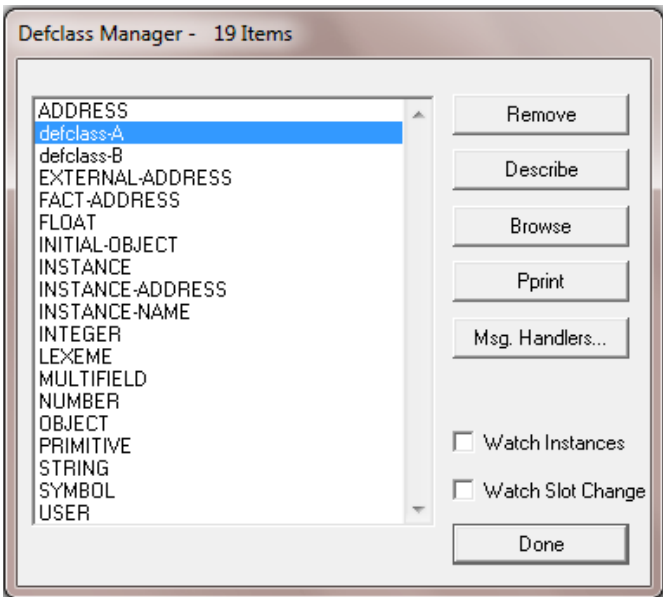

Рис. 1.22. Вигляд випадного вікна *Defclass Manager* меню *Browse.*

- Кнопка *Msg. Handlers* (див. рис.1.22) відкриває діалогове вікно, яке дозволяє переглядати обробники повідомлень для вибраного класу та виконати відповідну операцію на обробнику, обраному зі списку (рис. 1.23). Натискання кнопки Done повертає керування у діалогове вікно Defclass Manager.

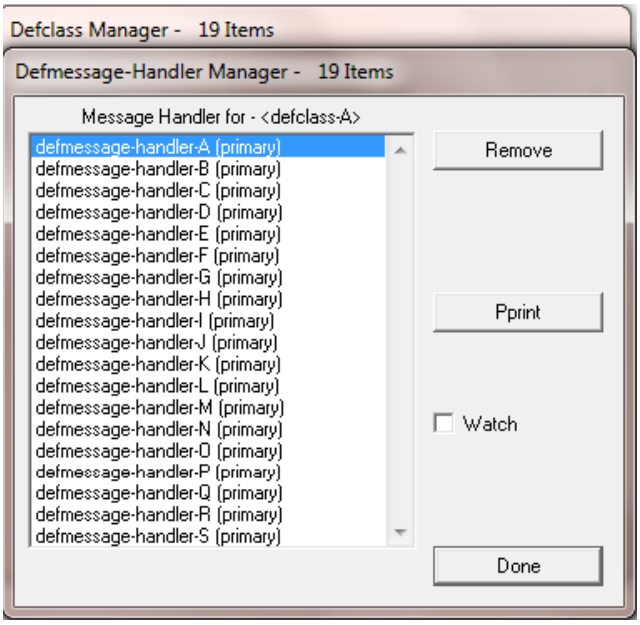

Рис. 1.23. Вигляд випадного вікна *Msg. Handlers.*

- Команда *Agenda Manager* відображає діалогове вікно (рис. 1.24), яке дозволяє перевірити активації в порядку денному і вилучити або запустити будь-яку вибрану зі списку.

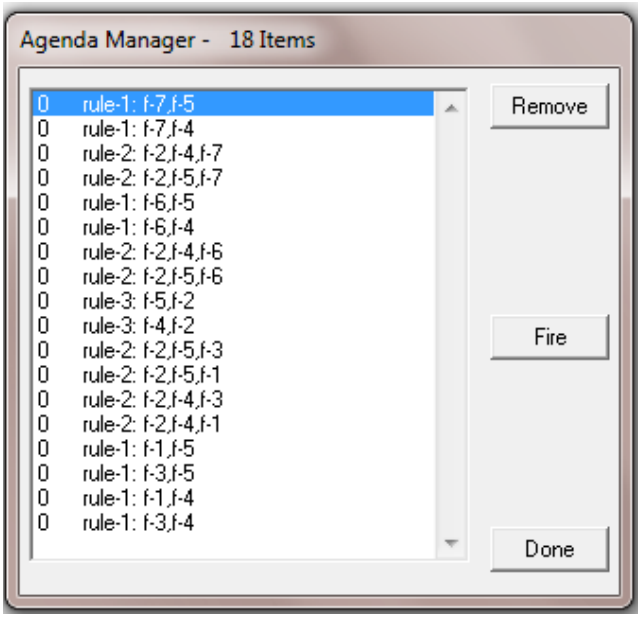

Рис. 1.24. Вигляд випадного вікна *Agenda Manager* меню *Browse.*

1.2.6 Меню Window.

Після натискання на кнопку *Window* у вікні CLIPS з’явиться випадне вікно цього меню (рис. 1.25).

- Перші чотири команди меню дозволяють розмістити вікна CLIPS у діалоговому вікні каскадно, горизонтально, зверху вниз або ж закрити всі.
- Команда *Show Status Windows* показує вікна Facts Window, Agenda Window, Instances Window, Globals Window, та Focus Window.
- Команда *Hide Status Windows* приховує вікно вказані вище вікна.
- Команда *Tile Dialog & Status Windows* розміщує вікна діалогу та статусу як непересічні об’єкти.

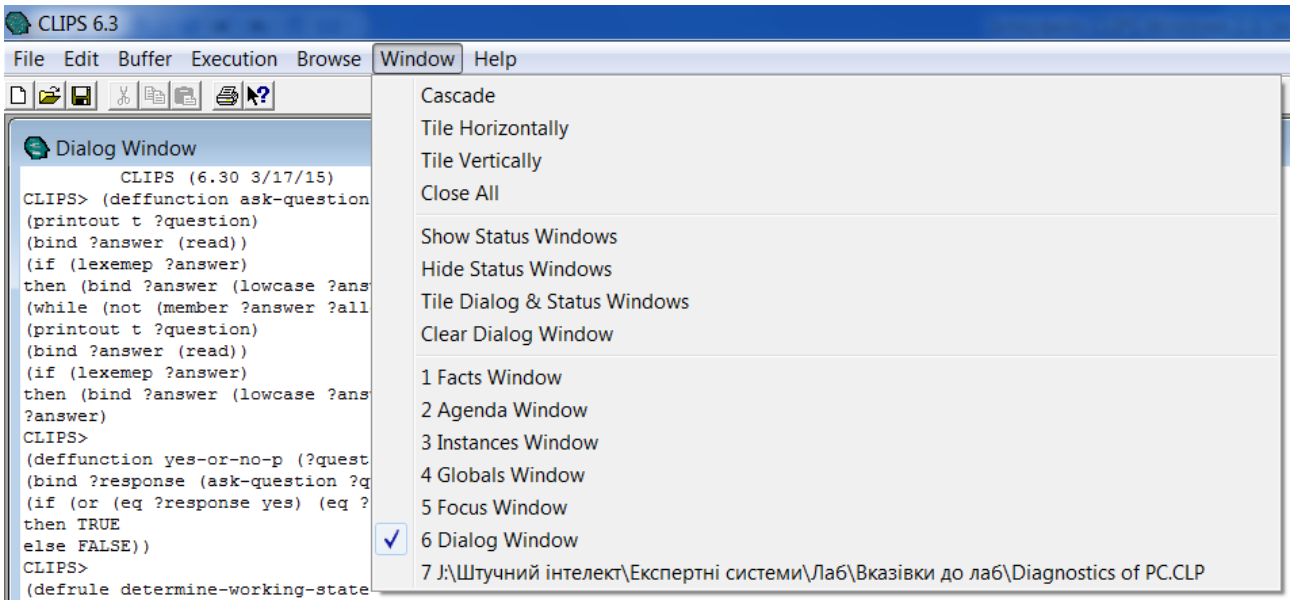

Рис. 1.25. Вигляд випадного вікна меню *Window*.

- Команда *Clear Dialog Window* очищає весь текст у вікні діалогу. З командного рядка CLIPS команда (clear-window), яка не містить аргументів,) також очистить весь текст у вікні діалогу.
- Команда *Facts Window* показує вікно Facts, у якому відображається список та кількість фактів у списку фактів. Перед іменем вікна розміщується прапорець, якщо це найголовніше вікно.
- Команда *Agenda Window* показує вікно Agenda, у якому відображається список активних правил, що знаходяться на порядку денному. Перед іменем вікна розміщується прапорець, якщо це найголовніше вікно.
- Команда *Instances Window* відображає вікно Instances, у якому відображається список існуючих екземплярів та значення їхніх слотів. Перед іменем вікна розміщується прапорець, якщо це найголовніше вікно.
- Команда *Globals Window* показує вікно Globals, у якому відображається список глобальних змінних та їх поточні значення. Перед іменем вікна розміщується прапорець, якщо це найголовніше вікно.
- Команда *Focus Window* відображає вікно фокусування. Вікно фокусу показує

поточний стек фокусу. Якщо це найголовніше вікно, перед його іменем розміщується прапорець,.

- Команда *Dialog Window* відкриває діалогове вікно CLIPS на передньому плані. Перед іменем вікна розміщується прапорець, якщо це найголовніше вікно.
- Якщо відкритий вбудований редактор, у меню *Window* з'являється команда під номером 7, натискання якої приводить до появи вінка редактора з його вмістом.

Окрім розглянутих, в інтерфейсі CLIPS присутнє меню довідки *Help* (рис. 1.26), використання якого дозволяє отримати потрібну користувачеві інформацію про середовище, у т. ч. щодо версії, яка використовується, а також надає доступ до домашньої інтернет-сторінки середовища, online-документації діючих версій CLIPS, різноманітних Інтернет-ресурсів, пов'язаних з висвітленням різних аспектів застосування CLIPS.

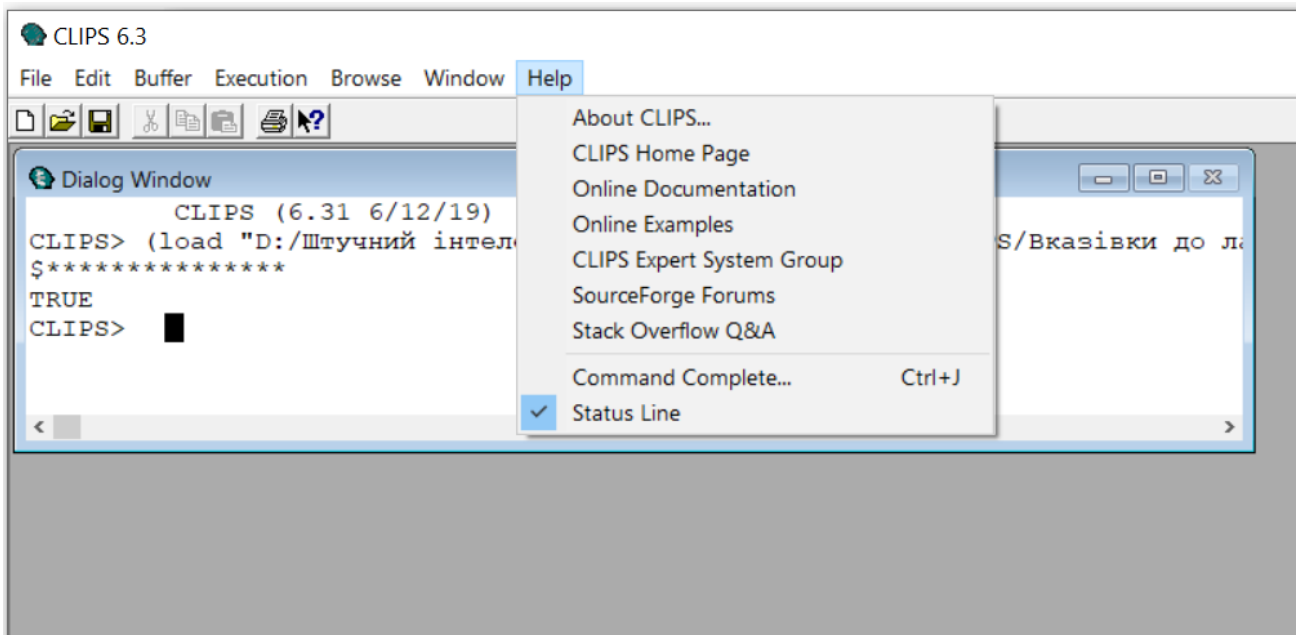

Рис. 1.26. Вигляд випадного вікна меню *Help*.

Порядок виконання роботи.

1. Ознайомитися з призначенням та основними властивостями мови CLIPS 6.31.
2. Встановити середовище CLIPS на комп'ютер. Для цього: зайти на сайт https://sourceforge.net/projects/clipsrules/files/CLIPS/6.30/, завантажити виконавчий файл (clips_windows_64_bit_executables_630.msi або ж clips_windows_32_bit_executables_630.msi, в залежності від розрядності ОС) з версією CLIPS 6.30 і встановити потрібну конфігурацію на ваш комп'ютер. Останню версію середовища можна встановити, завантаживши з ресурсу https://sourceforge.net/projects/clipsrules/files/CLIPS/6.31/ файл CLIPSIDE631LF та запустивши його.
3. Завантажити середовище CLIPS 6.30 для ОС Windows запуском файлу CLIPSIDE64.exe чи CLIPSIDE32.exe (версія 6.30 3/17/15 – в залежності від розрядності версії вашої ОС).
4. Ознайомитися з інтерфейсом середовища та призначенням основних пунктів меню віконного інтерфейсу.
5. Вивчити можливості використання віконного інтерфейсу для контролю за виконанням програм, зокрема – контролю введення фактів і правил та маніпуляцій з ними, трасування виконання програми та ін.
6. Ознайомитися з особливостями синтаксису написання команд CLIPS та режимами роботи середовища.
7. Освоїти особливості роботи в CLIPS з використанням командного рядка, вбудованого редактора та з використанням інших можливостей графічного інтерфейсу, у т. ч. запису програми у файл, завантаження її з файлу, тощо.
8. Написати будь-яку команду (наприклад, виведення на друк рядка якої-небудь інформації) та запустити її, використовуючи різні режими запуску – через командний рядок, з використанням вбудованого редактора, з записом у файл і введенням з записаного файлу.
9. Написати звіт про виконану роботу; при оформленні звіту використовувати

скріншоти окремих етапів роботи.

Контрольні запитання:

1. Що таке середовище CLIPS і для чого воно призначене?
2. До якого виду інструментальних засобів створення експертних систем можна віднести CLIPS?
3. Яка архітектура CLIPS і чим вона визначається?
4. Які переваги і недоліки використання CLIPS Ви можете назвати?
5. В чому полягають переваги Windows-версії CLIPS над його іншими версіями?
6. Що таке вбудований редактор CLIPS і які особливості його використання?
7. Які переваги надає можливість використання при написанні кодів команд і програм в CLIPS вбудованого редактора?
8. Що таке графічний інтерфейс CLIPS і в яких версіях середовища він володіє найкращими можливостями?
9. Які можливості надає CLIPS користувачеві в контролі виконання написаної програми?
10. Що має ввести в CLIPS користувач, щоб створити в ньому експертну систему?
11. Які парадигми програмування використовуються в CLIPS і за допомогою яких засобів вони реалізуються?
12. Який вид синтаксису використовується при створенні функцій та використання математичних операцій в CLIPS?
13. В чому полягає особливість використання синтаксису, застосованого в середовищі?
14. Як можна ввести код набраної в зовнішньому редакторі програми в CLIPS?
15. Які команди найчастіше використовуються при роботі CLIPS і в чому їх особливості?

Література:

1. CLIPS. A Tool for Building Expert Systems [Електронний ресурс], 2016. – Режим доступу: https://sourceforge.net/projects/clipsrules/files/CLIPS/6.30/clips_documentation_630.zip/download
2. CLIPS Rule Based Programming Language [Електронний ресурс], 2016. – Режим доступу: https://sourceforge.net/projects/clipsrules/files/CLIPS
3. CLIPS Reference Manual. Volume I. Basic Programming Guide. Version 6.30 March 17th 2015 – 416 p. [Електронний ресурс] – Режим доступу: http://clipsrules.sourceforge.net/documentation/v630/bpg.pdf
4. Джарратано Джозеф. Экспертные системы. Принципы разработки и программирование, 4е изд: Пер. с англ. / Джозеф Джарратано, Гари Райли. – М., Изд. дом “Вильямс”, 2007. – 1152 с.
5. Белкин Е.В. Введение в методы программных решений. Учебное пособие. / Е.В. Белкин, А. В. Гахов, А.М. Горбань, В.М. Куклин и др. – Х.: ХНУ имени В. Н. Каразина, 2010. – 305 с.
6. Частиков А. П. Разработка экспертных систем. Среда CLIPS. / А. П. Частиков., Т. А. Гаврилова, Д. Л. Белов. – СПб., «БХВ-Петербург», 2003. – 608 с.

Лабораторна робота 2.

Тема: ПРОГРАМУВАННЯ МАТЕМАТИЧНИХ ВИРАЗІВ У CLIPS. СТВОРЕННЯ ВЛАСНИХ ФУНКЦІЙ КОРИСТУВАЧА

Мета роботи: Освоїти можливості програмування математичних вираів, які надає мова програмування CLIPS.

Завдання до роботи:

- Ознайомитися з особливостями роботи в CLIPS в режимі процедурного програмування.
- Використовуючи діалогове вікно та вбудований редактор, провести обчислення деяких запропонованих викладачем математичних виразів.
- Створити нову користувацьку функцію CLIPS та провести розрахунки з її допомогою.

Вихідні відомості:

2.1. Синтаксис визначень.

Як базовий синтаксис для визначення конструкцій в CLIPS використовується стандартна БНФ-нотація (Бекуса-Наура нотація).

Для опису синтаксису різних команд і конструкцій CLIPS використовується система позначень, яка складається з трьох різних типів тексту, що підлягає введенню. Різниця між ними полягає у вигляді символів (дужок), в які поміщено відповідну конструкцію – (), [], <> чи {}. В залежності від того, у який вид з цих символів поміщений вираз або слово, говорять про *термінальні* (вирази, поміщені в (), [] і {}) або *нетермінальні* (вирази, поміщені в <>) *символи* CLIPS.

Позначення термінального типу відносяться до символів і знаків, які повинні

бути введені точно так, як вони показані у визначенні; до них відносяться будь-які текстові написи, поміщені всередину круглих дужок. Наприклад, опис синтаксису (example) означає, що конструкція (example) повинна бути введена так: спочатку – знак відкриваючої круглої дужки, потім букви напису example і, нарешті, знак закриваючої круглої дужки.

Квадратні дужки указують, що поміщений у них вміст є необов'язковим. Наприклад, опис синтаксису (example [1]) показує, що цифра 1, що знаходиться в квадратних дужках, може не вказуватися. Таким чином, результати введення (example) та (example 1) є сумісними з результатом введення синтаксичного виразу (example [1]).

Опис, в якому вводиться тип даних, який поміщений між знаками 'менше' і 'більше' ('<' і '>'), вказує, що повинна бути виконана заміна даних на певні значення того типу, який знаходиться усередині цих знаків. Наприклад, опис <integer>показує, що повинна бути виконана заміна усіх даних на дійсні цілочисельні значення. Так, опис синтаксису:

(example <integer>)

може бути замінено наступними результатами введення:

(example 1);

(example 8);

(example -2),

або будь-якими іншими результатами введення, в яких містяться знаки '(example ', за якими знаками слідує якесь ціле число, а за ним – знак ')'. Важливо відзначити, що пропуски, показані в описі синтаксису, також повинні бути включені в результат введення.

Нагадаємо ще раз, що слово або вираз, поміщені в кутові дужки '<>', називається *нетермінальним символом* (наприклад, <string>). Важливо при написанні кодів програм і команд пам'ятати, що будь-який нетермінальний символ вимагає обов'язкового подальшого визначення. Слова ж або вирази, поміщені не в кутові

дужки, називаються *термінальними символами* і представляють синтаксис описуваної конструкції мови CLIPS. Термінальні символи (особливо поміщені в круглі дужки) повинні вводитися в командний рядок саме так, як показано у їх визначенні.

Ще один варіант позначення в CLIPS характеризується використанням символу ‘*’, який без пробілу слідує за відповідним описом. Якщо за нетермінальним символом слідує символ ‘*’, то це означає, що в даному місці може знаходитися список з нуля або більше елементів цього типу. Наприклад, опис синтаксису:

<integer>*

може бути замінений одним з таких результатів введення:

1;

1 2;

1 2 3,

або будь-якою іншою кількістю розділених пропусками цілих чисел чи ж взагалі залишений порожнім.

Якщо ж за нетермінальним символом без пробілу слідує знак ‘+’, то це значить, що в цьому місці може знаходитися список з одного або більше елементів даного типу. Іншими словами, знак ‘+’, який слідує за описом, указує, що замість цього опису синтаксису повинне бути введене одне або декілька значень, заданих цим описом. Таким чином, опис синтаксису <integer>+ еквівалентний такому опису синтаксису, як <integer><integer>*.

Символи ‘*’ і ‘+’, що зустрічаються самі по собі (тобто не наступні після нетермінальних символів і відділені від них пробілами), є термінальними.

Двокрапка ‘:’, як і горизонтальна багатокрапка, також використовується для відображення списку з одного або більше елементів.

Елементи, поміщені в квадратні дужки (наприклад [<коментарі>]), є необов’язковими елементами, які можуть входити у визначення.

Вертикальна межа ‘|’, що розділяє два або більше елементів визначення,

вказує на те, що в конструкції необхідно використовувати один з перерахованих елементів, тобто вказує на необхідність вибору одного з декількох елементів, розділених вертикальними рисами. Наприклад, опис синтаксису:

all | none | some

може бути замінено таким результатом введення:

all;
none;
some.

Символ ::= використовується для позначення необхідності заміни деякого нетермінального символу (може читатися, наприклад, як ‘це значить’). Для прикладу, визначення:

<lexeme>: : = <symbol> | <string>

означає, що нетермінальний символ <lexeme>, що зустрічається в деякому визначенні, повинен бути замінений або на символ <symbol>, або на символ <string>.

Пропуски, символи табуляції, переходи на інший рядок використовуються тільки для логічного розділення елементів визначення й ігноруються CLIPS (окрім рядків, поміщених в подвійні лапки).

Загалом, при написанні програм у CLIPS синтаксис мови середовища можна розбити на такі три основні групи елементів:

- типи даних;
- функції, що використовуються для обробки даних;
- конструктори, призначені для створення таких структур мови, як факти, правила, класи тощо.

2.2. Команди і функції CLIPS та їх застосування.

Функцією в CLIPS називається частина коду, що має ім'я і повертає певний результат або виконує певні дії (наприклад, відображення інформації на екрані). Функції, що не повертають результат і які виконують деяку корисну роботу, зазвичай ще називаються *командами.*

Суттєвою різницею між ними є лише те, що термін 'функція' використовується для вказівки на те, що в результаті її виконання відбувається повернення деякого значення, а термін 'команда' – для того, щоб показати, що в результаті її виконання або не відбувається повернення значення, або відповідна дія зазвичай виконується тільки після введення даних. Тому, певним чином, терміни 'команда' і 'функція' в CLIPS слід розглядати як взаємозамінні.

При роботі в CLIPS найчастіше використовуються такі команди:

- *clear* – команда повного очищення робочої пам'яті системи – видаляє усі визначені в системі на даний момент конструктори й асоційовані з ними дані.
- *exit* – команда завершення сеансу роботи з CLIPS.
- *reset* – команда перезавантаження робочої пам'яті системи – очищає поточний план рішення задачі, видаляє всі факти зі списку фактів і об'єкти із списку об'єктів. При цьому в систему додається зумовлений факт *initial-fact*, зумовлений об'єкт *initial-object* і всі факти, об'єкти і глобальні змінні, визначені користувачем за допомогою конструкторів *deffacts*, *definstances* і *defglobals*.
- *printout* – команда виведення даних.
- *run* – команда запуску CLIPS.

2.3. Програмування математичних операцій у CLIPS.

Будь-який символ, поміщений в круглі дужки, розглядається CLIPS як ко-

манда або ж як виклик функції. Наприклад, результатом введення виразу (+ 3 4) стає виклик функції додавання, яка здійснює складання вказаних двох чисел, оскільки першим у виразі стоїть позначення функції додавання.

Позначення деяких з передбачених у CLIPS стандартних арифметичних і математичних функцій показані у табл. 2.1. Докладніша інформація про деякі часто вживані функції CLIPS надана в [1].

Таблиця 2.1. Запис деяких математичних функцій в CLIPS.

Функція	Позначення функції в CLIPS
Додавання	+
Віднімання	-
Множення	*
Ділення	/
Піднесення до ступеня	* *
Визначення абсолютного значення	abs
Обчислення квадратного кореня	sqrt
Цілочисельне ділення	div
Залишок від ділення	mod
Знаходження мінімуму	min
Знаходження максимуму	max
Синус	sin
Косинус	cos
Тангенс	tan
Натуральний логарифм	log
Експонента e^x	exp
Округлення числа	round
Вибір цілого випадкового числа з інтервалу $[n_1, n_2]$	random n_1 n_2

При здійснення функціональних обчислень потрібно пам’ятати:

- кожна команда CLIPS повинна мати погоджену кількість відкриваючих и закриваючих круглих дужок;
- будь-які символи, не поміщені в дужки, інтерпретатором CLIPS сприймаються як константи.

При виклику функції в CLIPS потрібно враховувати префіксну форму їх представлення – аргументи функції можуть стояти тільки після її назви. Виклик функції починається з дужки, що відкривається, за якою слідує ім’я функції; потім слідують аргументи, кожен з яких відокремлений одним або декількома пробілами. Аргументами функції можуть бути дані простих типів, змінні або ж виклики інших функцій. В кінці виклику ставиться дужка, що закривається. Наприклад, код для обчислення вираз 3+8×12+7 в CLIPS записується таким чином:

(+ 3 (* 8 12) 7),

а виразу $\cos(4\ln(e^{2\sin 2} + 2^3))$ – як

(cos (* 4 (log (+ (exp (* 2 (sin 2))) (** 2 3))))).

Також, при програмуванні складних математичних виразів, які вміщують декілька різних операцій, їх потрібно групувати за рангом – першою у коді повинна стояти операція, яка є ‘головною’, тобто зв’язує всі частини виразу.

Таким чином, використовуючи синтаксис опису CLIPS та відповідні стандартні арифметичні та інші математичні функції, можна здійснити обчислення математичного виразу практично будь-якої складності.

Отже, CLIPS дозволяє не тільки обробляти символічні факти, але і проводити обчислення. Але завжди слід враховувати, що мова створення експертних систем CLIPS, загалом, не призначена для проведення складних математичних розрахунків. Безумовно, математичні можливості CLIPS як засобу створення експертних систем є досить потужними, але вони, головно, призначені забезпечувати модифікацію тих числових значень, які застосовуються для здійснення операцій

логічного виведення за допомогою прикладних програм.

2.4. Змінні та групові символи у CLIPS.

Як і в інших мовах програмування, в CLIPS для зберігання значень використовуються змінні, які, згідно особливостей свого використання, можуть бути локальними (чинними тільки в межах певного модуля, в якому вони визначені) або глобальними (які чинні у всьому середовищі CLIPS). Спільним для обох видів змінних є те, що в своєму синтаксисі вони обов'язково повинні використовувати знак '?'.

Локальні змінні

Локальні змінні CLIPS записуються як:

?<variable-name>

Ідентифікатор змінної завжди починається зі знака '?', за яким (без пробілу) слідує її ім'я (ім'я поля або слота факту) типу <symbol>.

Приклади змінних: ?x, ?x1, ?name, ?noun, ?color.

Усі змінні, окрім глобальних, вважаються локальними і можуть використовуватися тільки в рамках опису текучої конструкції (модуля); до локальних змінних можна звертатися тільки всередині опису, але вони не визначені поза ним.

Перед використанням змінної їй необхідно привласнити значення.

Зазвичай змінні в CLIPS описуються і отримують свої значення в лівій частині правила. Отримавши значення, змінна зберігає його незмінним при використанні як в лівій, так і в правій частині правила, якщо тільки це значення не змінюється в правій частині за допомогою функції *bind*.

Крім значення самого факту, змінній також може бути присвоєне і значення адреси факту. Це може виявитися зручним при необхідності маніпулювати фактами безпосередньо з правила – наприклад, для того, щоб визначити, який факт

буде змінюватися, необхідно привласнити змінній адресу конкретного факту. Для такого присвоєння використовується комбінація знаків ‘<-’. Присвоєння адрес відбувається в лівій частині правила з синтаксисом:

<pattern-address> :: = ?<name-variable> <- <pattern>;

отримане значення називається адресою зразка (pattern-address).

Стрілка ‘<-’ – необхідна частина синтаксису присвоєння. Змінна, пов’язана з адресою факту або об’єкта, може порівнюватися з іншою змінною або використовуватися зовнішньої функцією. Змінна, пов’язана з адресою факту, також може бути також використана для подальшого обмеження полів в зразку умовного виразу. Однак, потрібно пам’ятати, що не можна пов’язувати змінну в умовному елементі *not*.

Після зв’язування цю зміну можна використовувати в командах правої частини правила – *retract*, *modify* або *duplicate* замість індексу факту.

Глобальні змінні

Для визначення глобальних змінних, які, на відміну від локальних, чинні всюди (тобто у будь-якому модулі програми середовища CLIPS), використовується конструкція *defglobal* з синтаксисом:

(defglobal [<defmodule-name>] <global-assignment>*).

У цьому визначенні параметр <global-assignment> задається наступним чином:

<global-assignment> ::= <global-variable> = <expression>,

а параметр <global-variable> визначається як:

<global-variable> ::= ?*<symbol>*

Необов’язковий терм <defmodule-name> представляє собою модуль, в якому

повинні бути визначені глобальні змінні – якщо цей параметр не заданий, то глобальні змінні поміщаються в поточний модуль.

Імена глобальних змінних починаються і закінчуються знаком ‘*’, завдяки чому можна легко відрізнити локальну змінну, що позначається як ?*x*, від глобальної, яка позначається як ?**x**. Як видно з визначення конструкції *defglobal,* початкове значення для кожної глобальної змінної задається шляхом обчислення виразу <expression> і присвоювання отриманого значення змінній *defglobal*.

До глобальної змінної можна звернутися в будь-якому місці і в будь-якій частині будь-якого правила; її значення залишається незалежним від інших конструкцій. Глобальні змінні CLIPS подібні до глобальних змінних у процедурних мовах програмування – з тією відмінністю, що на них не накладається обмеження на зберігання даних тільки одного типу.

Глобальні змінні можуть використовуватися в будь-якому місці, де може використовуватися локальна змінна; вони не можуть використовуватися як змінний параметр для конструкторів *deffunction*, *defmethod* або в оброблювачі повідомлень.

Маніпулювати конструкціями *defglobal* (вивести на зовнішній пристрій її текстове представлення, видалити, відобразити список, імена і значення глобальних змінних) можна за допомогою команд з синтаксисом:

(get-defglobal-list [<module-name>])
(list-defglobals [<module-name>])
(show-defglobals [<module-name>])
(ppdefglobal <defglobal-name>)
(undefglobal <defglobal-name>)

Функція *get-defglobal-list* повертає багатозначне значення, яке містить список конструкцій *defglobal*.

Команда *list-defglobals* призначена для відображення в діалоговому вікні

списку імен всіх визначених у системі глобальних змінних. Якщо необов'язковий параметр <module-name> не вказаний, то дана команда виводить імена глобальних змінних, визначених у поточному модулі. Якщо ж він містить ім'я конкретного модуля, команда *list-defglobal* виводить список змінних, визначених у заданому модулі. У випадку використання як параметра символу '*' команда виведе в діалогове вікно імена всіх глобальних змінних, визначених у всіх модулях системи.

Команда *show-defglobals*, на відміну від команди *list-defglobals*, виводить у діалогове вікно CLIPS не тільки імена глобальних змінних, але і їх значення. В іншому ці дві команди практично ідентичні.

Потрібно пам'ятати, що як <defglobal-name> для команд *ppdefglobal* і *undefglobal* повинно використовуватись ім'я глобальної змінної без початкових і кінцевих знаків '*' (наприклад, *x*, а не **x**).

Приклад визначення глобальних змінних та застосування до них вказаних функцій показані на рис. 2.1.

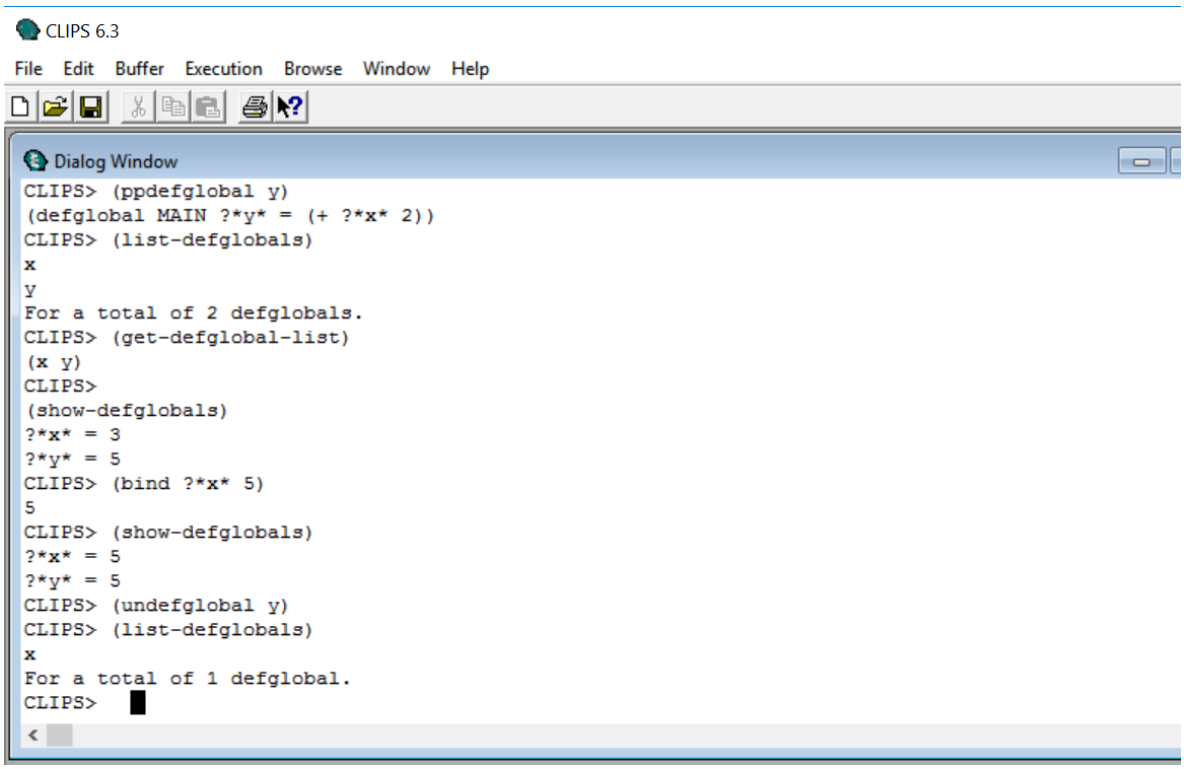

Рис. 2.1. Визначення глобальних змінних та результати застосування функцій для маніпуляцій з ними.

Змінити значення заданої змінної (чи створити її) можна за допомогою функції *bind* з синтаксисом:

(bind <variable> <value>*)

де під <variable> у нашому випадку розуміється ім’я глобальної змінної.

2.5. Функції та процедурні можливості CLIPS.

Поряд з евристичною парадигмою подання знань (яка забезпечується їх представленням у вигляді правил, і, згідно призначення, є визначальною для CLIPC), у середовищі при створенні експертних систем передбачені можливості для ефективного застосування і процедурної парадигми (забезпечується використанням процедурних функцій). Однак, з погляду на те, що мова CLIPS призначена головно для використання в якості ефективної мови, що ґрунтується на правилах, процедурні функції передбачені головно лише для обмеженого застосування, насамперед – для забезпечення використання обмежень та порівнянь в лівій та правій частині правил. Найчастіше процедурні функції використовуються для виконання простих перевірок і циклів головним чином у правій (консеквентній) частині правил. Загалом, виходячи з загального призначення середовища, заснованого на правилах, бажано уникати застосування у їх правих частинах складних вкладених конструкцій, що містять процедурні функції.

Для забезпечення процедурної парадигми програмування в CLIPS використовуються вбудовані (внутрішні) та зовнішні (користувацькі) функції; під функціями тут розуміється послідовність дій, описаних в конструкції з певним іменем, які повертають деяке значення або виконують інші задані дії.

Внутрішні функції (їх ще називають системними) вбудовані в середовище CLIPS при його створенні і їх можна використовувати в будь-який момент роботи. Вони використовуються для організації роботи програм (функції *run, clear, reset, save, exit* тощо) та для керування потоком виконання дій у середовищі

CLIPS (*while, if, switch, loop-for-count, prong$, break;* а також функція *halt*, яка, задана в правій частині правила, дозволяє зупинити його виконання).

Зовнішні функції – це функції, написані користувачем спеціально для задоволення певних процедурних проблем, які виникають при створенні програми; створювати їх можна як за допомогою середовища CLIPS, так і з використанням інших мов програмування з наступним підключенням вже готових, відкомпільованих виконуваних модулів до CLIPS. На відміну від внутрішніх, зовнішні функції виконуються не напряму, а інтерпретуються середовищем.

Для створення нових функцій в CLIPS використовується конструктор *deffunction*.

2.3. Конструктор *deffunction* та особливості створення і застосування створених функцій.

Конструктор *deffunction* дозволяє користувачеві створювати нові функції безпосередньо в середовищі CLIPS. При її створенні повинні виконуватися такі вимоги:

- конструктор *deffunction* повинен бути оголошений до першого використання функції, ним створеної;
- функція, що створюється за допомогою конструктора *deffunction*, повинна мати унікальне ім’я, що не збігається з іменами інших зовнішніх і внутрішніх функцій;
- функція, створена з допомогою *deffunction*, є простою, і, на відміну від т. з. родових функцій, не може бути перевантаженою; іншими словами, вона може виконувати тільки приписані їй дії одного типу.

Синтаксис конструктора deffunction має вигляд:

```
(deffunction <deffunction-name>
[<optional-comment>]
```

(<regular-parameter>*)
[<wildcard-parameter>]
(<expression>*)
)

і включає в себе 5 елементів:

- ім’я функції <deffunction-name>;
- необов’язкові коментарі [<optional-comment>];
- список з нуля або більше параметрів (<regular-parameter>*);
- необов’язковий символ групових параметрів [<wildcard-parameter>] для вказівки того, що функція може мати змінне число аргументів;
- послідовність дій або виразів (<expression>*), які будуть виконані (обчислені) за порядком після виклику функції.

Ім’я конструкції *deffunction* повинне бути унікальним, зокрема, не повинне збігатися з іменем жодної зі вже існуючих, визначених до її створення функцій CLIPS.

Параметр <regular-parameter> представляє собою однозначну змінну, а <wildcard-parameter> – багатозначну змінну. Оголошення <regular-parameter> і <wildcard-parameter> дозволяє задавати параметри, що передаються в конструкцію *deffunction* при її виклику. Такий спосіб виклику до певної міри аналогічний до активування правил, в яких змінні, пов’язані зі значеннями в їх лівій частині, розглядаються як оголошення параметрів, що застосовуються в правій частині правила. Крім того, в тілі конструкції *deffunction* для створення локальних змінних може використовуватися команда *bind*, у повній відповідності з тим, як ця дія виконується в правій частині правила.

Кількість параметрів, якими оперує створена за допомогою конструктора *deffunction* функція, може приймати або точне число, або ж число, не менше від деякого заданого – залежно від того, встановлено груповий параметр чи ні. Обов’язкові параметри визначають мінімальне число аргументів, яке має бути пе-

редано функції при її виклику. В діях функції можна посилатися на кожен з цих параметрів як на звичайні змінні, що містять прості значення. Якщо ж було задано груповий параметр, то створена функція може приймати будь-яку кількість аргументів, але їх число має бути більшим або ж рівним мінімальному числу. Якщо груповий параметр не заданий, то функція може приймати число аргументів, що точно дорівнює кількості обов'язкових параметрів. Всі аргументи функції, які не відповідають обов'язковим параметрам, групуються в одне значення складеного поля. Посилатися на це значення можна, використовуючи символ групового параметра. Визначення функції може містити лише один груповий параметр.

Тіло конструкції *deffunction*, представлене параметром <expression>* і, подібно до правої частини правил, складається з низки виразів, які виконуються послідовно після виклику створеної з її допомогою фцнкції.

Визначені користувачем конструкції *deffunction* діють аналогічно до заздалегідь визначених (системних) функцій, передбачених у мові CLIPS. В будь-яких умовах, що допускають виклик заздалегідь певної системної функції CLIPS, можна викликати і конструкцію *deffunction*. Але, на відміну від заздалегідь визначених (системних) функцій, допускається видалення конструкцій *deffunction*, а для відстеження їх виконання може використовуватися команда *watch*.

Конструкція *deffunction* може повертати значення, за аналогією з тим, як повертають значення деякі наперед визначені функції. Її обчисленим значенням є значення останнього виразу, обчисленого в тілі конструкції.

Спосіб виклику функцій, визначених користувачем, еквівалентний способу виклику внутрішніх функцій CLIPS – її виклик здійснюється за іменем, заданим користувачем. За ім'ям функції слідує список необхідних аргументів, відокремлений одним або більшим числом пропусків. Виклик функції разом зі списком аргументів повинен бути поміщений у дужки.

Послідовність дій визначеної з допомогою конструктора *deffunction* функції виконується інтерпретатором CLIPS (на відміну від функцій, створених на інших мовах програмування, які повинні мати вже готовий виконавчий код).

Приклад використання створеної за допомогою конструктора *deffunction* функції для визначення довжини гіпотенузи прямокутного трикутника:

```
CLIPS> (deffunction hypotenuse-length (?a ?b)
(bind ?temp (+ (* ?a ?a) (* ?b ?b)))
(return (** ?temp 0.5)))
CLIPS> (hypotenuse-length 2 3)
3.60555127546399
CLIPS> (hypotenuse-length 3 4)
5.0
CLIPS>
```

2.4. Процедурні функції CLIPS, що використовуються для організації циклів та галуження

CLIPS підтримує такі процедурні функції, які використовуються для створення блоків процедурного програмування та надають можливості щодо реалізації в програмах:

if – галуження;
while – циклу з передумовою;
loop-for-count – ітеративного циклу;
switch – множинного галуження;
prong – об’єднання дій в одній логічній команді;
prongs – виконання набору дій над кожним елементом поля;
return – переривання функції, циклу, правила тощо;
break – аналог *return*, але без повернення параметрів;
bind – створення та зв’язування змінних.

Функція if.

Функція *if* має такий синтаксис:

(if <predicate-expression> then <expression>+ [else <expression>+]).

У цьому визначенні <predicate-expression> – це один вираз (предикативна функція або змінна), а параметр <expression>+, який слідує за ключовими словами *then* і *else*, являє собою один або декілька виразів, які повинні бути обчислені (або виконані) з урахуванням значення, отриманого в результаті обчислення виразу <predicate-expression>. Частина функції else <expression>+ є необов'язковою.

Дія функції полягає в наступному.

При виконанні функції *if* спочатку перевіряється умова, представлена у виразі <predicate-expression> для визначення того, чи повинні бути виконані дії, задані в конструкції *then* або *else*. Якщо в результаті цієї перевірки отримується будь-який результат, відмінний від FALSE, то далі виконуються дії, задані в конструкції *then* цієї функції; в протилежному випадку виконуються дії, задані в конструкції *else*. Якщо ж конструкція *else* відсутня в тілі функції, то після отримання помилкових результатів перевірки ніякі дії не виконуються.

Обчисленим значенням функції *if* є результат обчислення останнього виразу в частині *then* або *else* функції. Якщо результатом обчислення виразу <predicate-expression> стає FALSE і у функції відсутня частина *else*, то функція *if* повертає символ FALSE.

Після завершення виконання функції *if* система CLIPS переходить до виконання наступної дії в тій частині правила, в яку вона включена (якщо воно є).

Функцію *if* зручно використовувати для перевірки значень в правій частині правила, оскільки це дозволяє позбутися від необхідності здійснювати перевірку з використанням інших правил.

Наприклад, за допомогою правила *continue-check*:

```
(defrule continue-check
```

```
?phase <- (phase check-continue)
=>
(retract ?phase)
(printout t ‘Continue? ‘)
(bind ?answer (read))
(if (or (eq ?answer y) (eq ?answer yes))
then (assert (phase continue))
else (assert (phase halt))))
```

можна визначити, чи повинне бути продовжено виконання програми. Якщо в CLIPS введено факт (phase check-continue), то середовище, в залежності від введеної відповіді (*yes* чи *no*) на відповідний запит, або продовжить (введе в список фактів новий факт (phase continue)), або зупинить (введе факт (phase halt)) виконання програми (рис. 2.2).

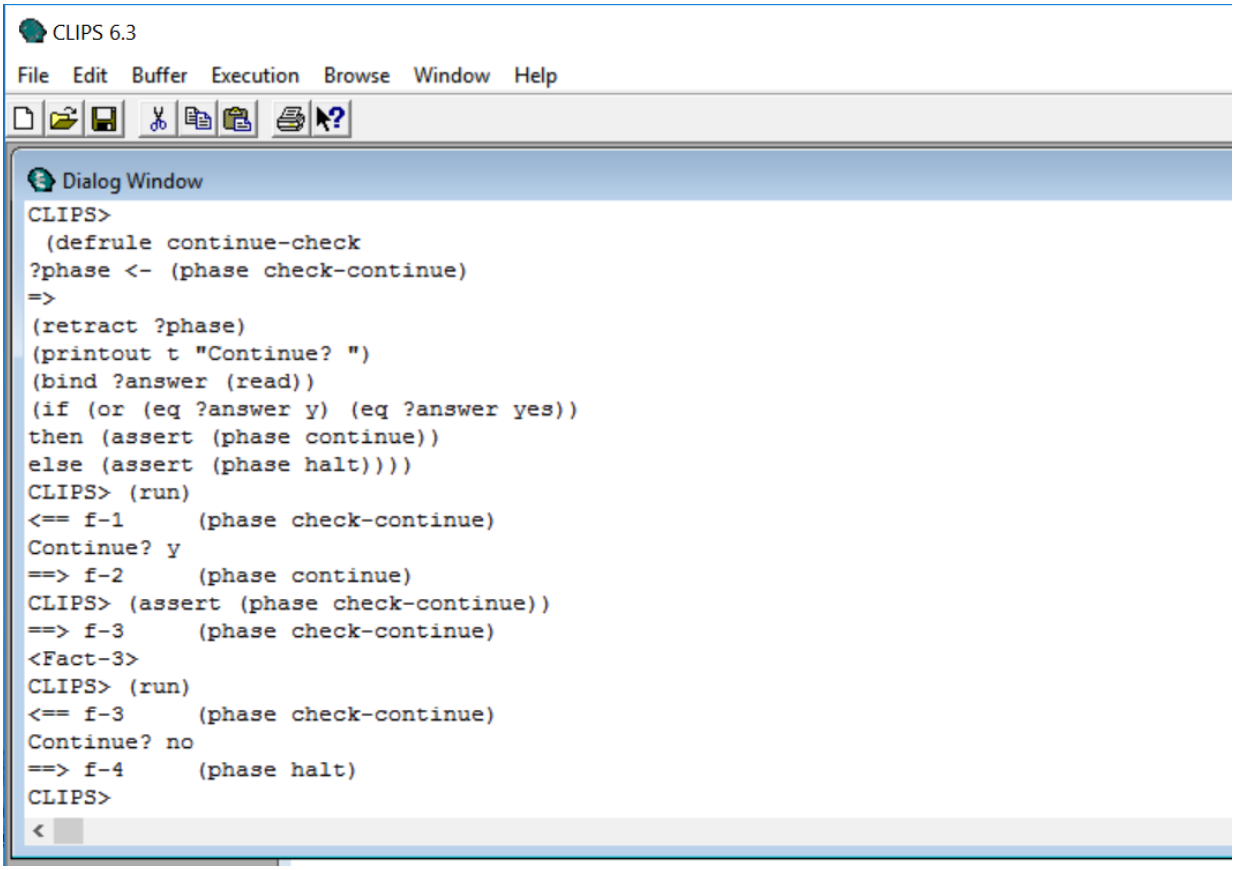

Рис. 2.2. Результати виконання правила defrule continue-check.

Слід зазначити, що функція *if* дозволяє перетворити позитивну або негативну відповідь, *yes* або *no*, у факт, який вказує, якого типу дію повинно бути виконано. У розглянутому випадку такою дією може бути або *continue*, або *halt*.

Функція while

Функція *while* дозволяє виконувати простий цикл і має такий синтаксис:

(while <predicate-expression> [do] <expression>*)

Як і в функції *if*, <predicate-expression> тут може бути представлена будь-якою предикативною функцією або змінною, а параметр <expression>*, позначає будь-яку кількість виразів (у т. ч. і включені цикли чи функцію *if*), які повинні бути обчислені з урахуванням значення, що повертається в результаті обчислення виразу <predicate-expression>. Ці вирази складають тіло циклу.

Частина функції *while*, яка представлена виразом <predicate-expression>, обчислюється до виконання дій, передбачених у тілі циклу. Якщо в результаті обчислення цього виразу буде отримане значення, відмінне від FALSE, то виконуються вирази, представлені в тілі циклу. Якщо ж результатом є FALSE, то програма переходить до виконання оператора, який слідує за функцією *while* (якщо такий є). Умова функції *while* перевіряється кожен раз перед виконанням операторів у тілі циклу для визначення того, чи повинні вони бути знову виконані.

Функція *while* може використовуватися разом з функцією *if* для перевірки помилок введення в правій частині правила. Нижче показаний приклад правила continue-check, в якому застосовується цикл для визначення умови продовження циклу до тих пір, поки не буде отримана прийнятна відповідь.

```
(defrule continue-check
?phase <- (phase check-continue) =>
(retract ?phase)
(printout t ‘Continue? ‘)
(bind ?answer (read))
```

(while (and (neq ?answer yes) (neq ?answer no)) do

(printout t ‘Continue? ‘)

(bind ?answer (read)))

(if (eq ?answer yes)

then (assert (phase continue))

else (assert (phase halt))))

Результати виконання CLIPS цього правила при введенні різних варіантів відповіді на поставлене запитання показані на рис. 2.3. Як бачимо, при введенні некоректної відповіді відбувається циклічне повторення запиту аж до введення відповіді у потрібному форматі (у нашому випадку – *no*), після чого виконується зумовлена нею дія.

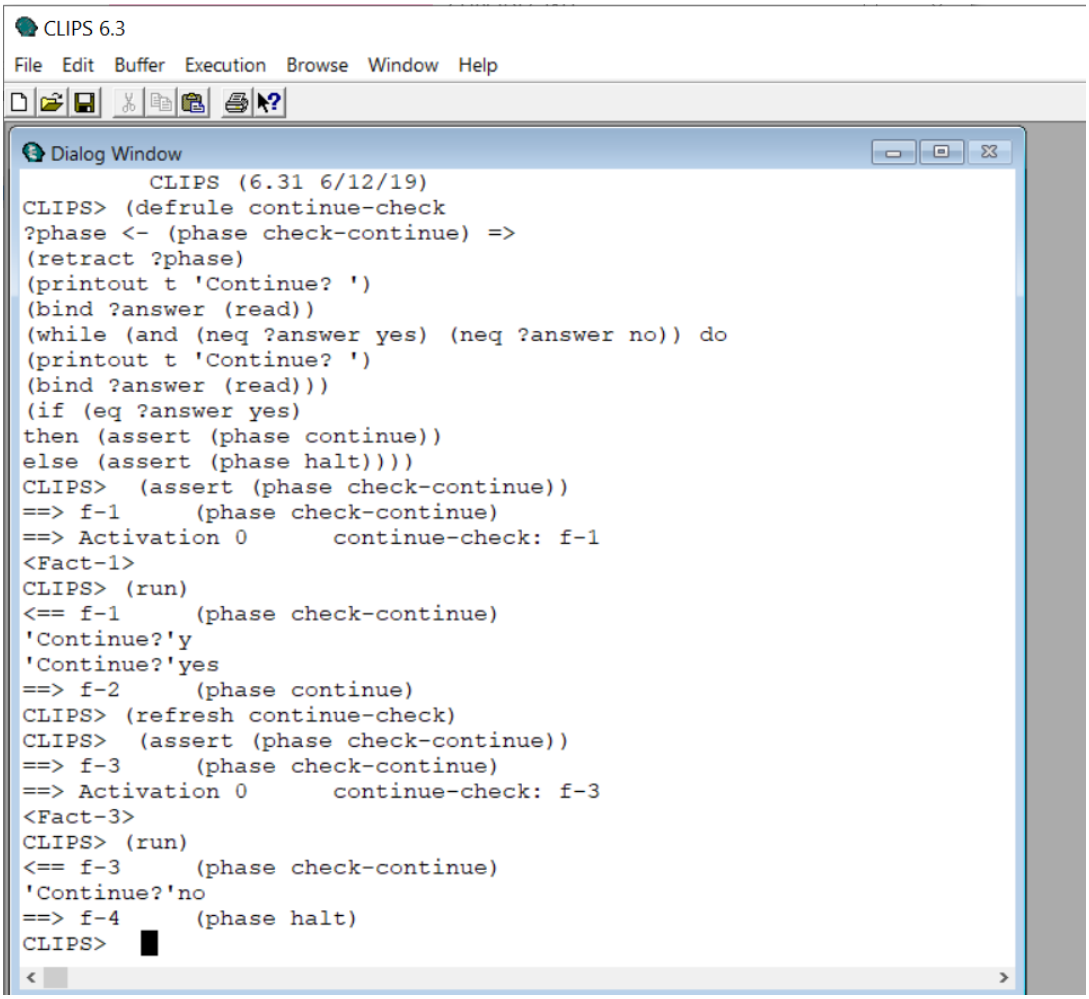

Рис. 2.3. Результат виконання правила continue-check з використанням функції *while*.

Функція loop-for-count.

Функція *loop-for-count* має синтаксис:

(loop-for-count <range-spec> [do] <expression>*).

і дозволяє реалізувати концепцію простого ітеративного циклу, що виконує дії циклу задане число разів.

У цьому визначенні <range-spec> означає кількість виконуваних дій, які задаються тілом функції <expression>*, і може набувати виду:

<range-spec> ::= <end-index> | (<loop-variable><end-index>) | (<loop-variable><start-index><end-index>)

Тут <start-index> і <end-index> являють собою цілі числа або ж вирази, обчислення яких дає цілочисельні значення. Якщо параметр <start-index> не заданий, йому автоматично присвоюється 1. Якщо заданий тільки вираз <end-index>, то його значення розглядається як кількість ітерацій виконання тіла функції.

Якщо задані вирази <loop-variable> і <end-index>, то здійснюється вказана кількість ітерацій виконання тіла функції, а поточний номер ітерації, в межах від 1 до <end-index>, зберігається у змінній <loop-variable> на кожній ітерації.

У випадку, коли, окрім <loop-variable> і <end-index>, задано і вираз <start-index>, то відлік кількості ітерацій починається не з 1, а зі значення <start-index>, і кількість виконуваних ітерацій визначається як різниця між значеннями <end-index> і <start-index> плюс одиниця. Якщо значення <start-index> більше від <end-index>, то ніякі ітерації не виконуються.

Для екстреного переривання роботи циклу можуть бути використані функції *break* і *return*. Тіло циклу може містити довільну кількість дій, у т. ч. включаючи й вкладені цикли або ж функцію *if*.

Функція *loop-for-count*, якщо не відбулося екстреного переривання її роботи, повертає значення FALSE.

Розглянемо приклад роботи функції *loop-for-count*, в якому використовуються всі варіанти застосування трьох складових конструкції <range-spec>. Для цього наберемо і введемо в CLIPS наступні функції:

```
(loop-for-count  2
(printout  t  'Hello!'  crlf))
 (loop-for-count  (?x  3)  do
 (loop-for-count  (?y  5  7)  do
(printout  t 'x=' ?x  '  y='  ?y '  x/y=' (/ ?x ?y) crlf)))
```

Результат виконання вказаних функцій в CLIPS показаний на рис. 2.4.

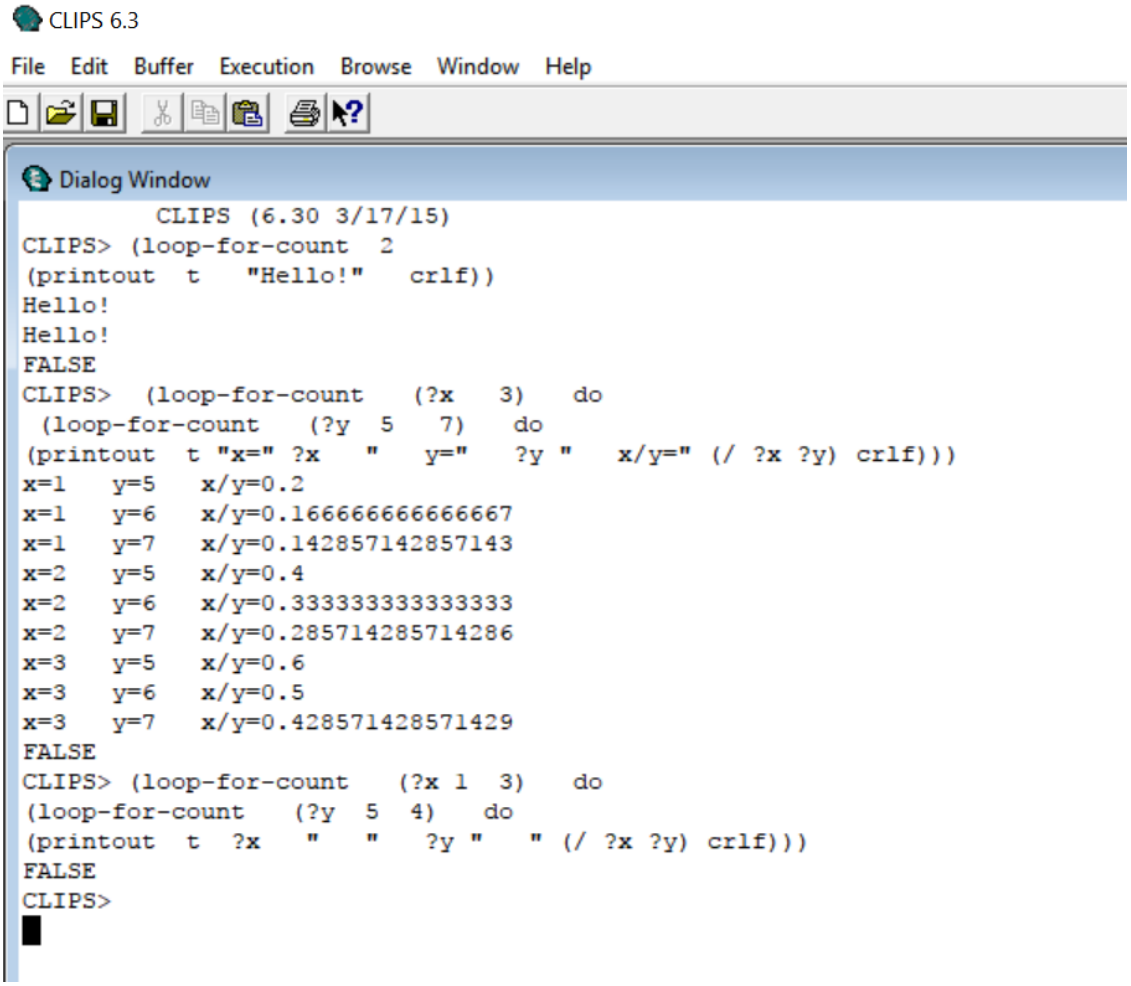

Рис. 2.4. Приклади виконання функції *loop-for-count* з різними значеннями її параметрів.

Функція switch

Функція *switch* дозволяє реалізувати множинне галуження та зв'язати певну групу дій (серед декількох подібних) з деякою заданою величиною і має синтаксис:

(switch <test-expression><case-statement>* [<default-statement>]),

де:

<case-statement>::= (case <comparison-expression> then <expression>*);

<default-statement>::= (default <expression>*).

Для ефективного застосування функції необхідна наявність в її тілі декількох (принаймні – трьох) альтернативних груп дій, що залежать від заданого умовного виразу.

Після передачі управління функції *switch* спочатку обчислюється частина, представлена виразом <test-expression>. Результат обчислення цього виразу порівнюється з результатом обчислення виразу <comparison-expression> кожної конструкції *case*. Якщо виявиться, що значення якоїсь конструкції *case* збігається з обчисленим значенням <test-expression>, то тоді обчислюється вираз <expression>*, що слідує після ключового слова *then* тільки цієї конструкції, і робота функції *switch* на цьому завершується.

Якщо ж такого збігу не виявляється і в тілі функції *switch* задана конструкція <default-statement>, то тоді виконується вираз <expression>* саме цієї конструкції з поверненням відповідного результату як результату роботи функції.

Частини конструкції <expression>*, які слідують за ключовими словами *then* і *default*, можуть складатися з одного або декількох виразів; слід також враховувати, що необов'язкова частина конструкції <default-statement> повинна бути розташована після всіх конструкцій *case* тіла функції *switch*.

Приклад використання функції *switch* в тілі правила, призначеного для встановлення відповідності між символічними іменами і знаками арифметичних функцій:

```
(defrule perform-operation
(operation ?type ?argl ?arg2)
=>
(switch ?type
(case times then
(printout t ?argl ‘ times ‘ ?arg2 ‘ is ‘ (* ?argl ?arg2)’.’ crlf))
(case plus then
(printout t ?argl ‘ plus ‘ ?arg2 ‘ is ‘ (+ ?argl ?arg2)’.’ crlf))
(case minus then
(printout t?argl ‘ minus ‘ ?arg2 ‘ is ‘ (- ?argl ?arg2)’.’ crlf))
(case divided_by then
(printout t?argl ‘ divided by ‘ ?arg2 ‘ is ‘ (/ ?argl ?arg2)’.’  crlf))))
```

Введення факту

```
(assert (operation plus 3 4))
```

та наступний запуск правила приведе до отримання результату:

```
CLIPS>  (assert (operation plus 3 4))
<Fact-1>
CLIPS> (run)
3 plus 4 is 7 .
CLIPS>
```

Символ *plus* у факті *operation* викликає перехід в операторі *switch* до варіанту *case* зі значенням *plus*, після чого виводиться результат додавання чисел 3 і 4.

Результати виконання правила *perform-operation* з послідовним введенням аналогічних до наведеного вище фактів з заміною значення *plus* на *minus*, *times* та *divided_by* показані на рис. 2.5.

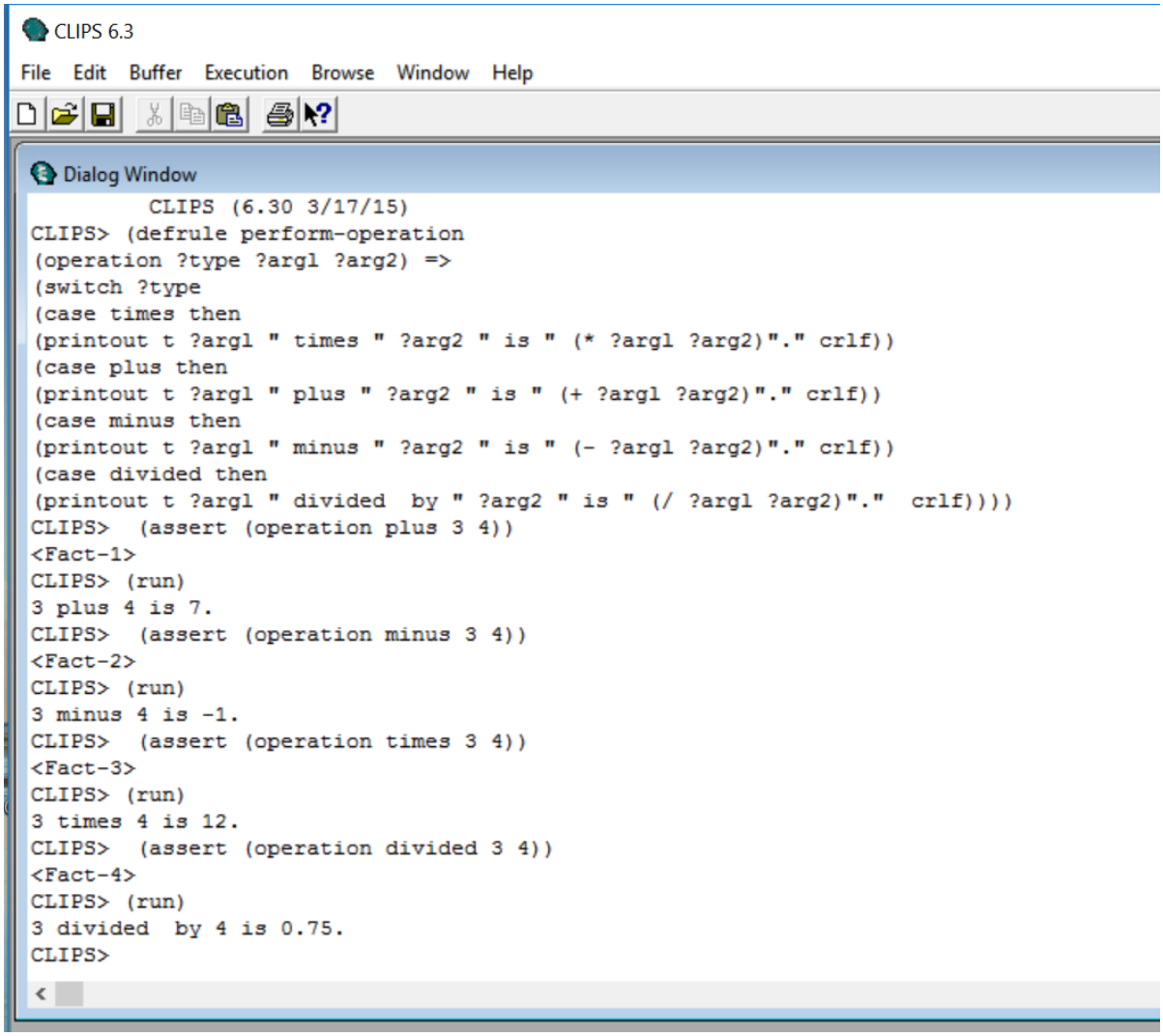

Рис. 2.5. Результати виконання правила *perform-operation* з оператором *switch*.

Функції progn і progn$

Функція *progn* з синтаксисом

(progn <expression>*)

дозволяє групувати і об’єднувати набір визначених дій в одну логічну команду і призначена для виконання кількох обчислень (дій) в рамках однієї команди. Функція виконує всі дії, задані виразом <expression> *), і повертає результат виконання останньої його дії.

Функція *progn*$, на відміну від progn, призначена для виконання заданого

набору дій над кожним елементом складеного поля, і має синтаксис:

(progn$ <list-spec><expression>*),

де параметр <list-spec> задається в такій формі:

<list-spec> ::= <multifield-expression> | (<list-variable><multifield-expression>)

Якщо в <list-spec> заданий параметр <multifield-expression>, то тіло функції, представлене як <expression>*, виконується по одному разу для кожного поля з результуючого багатозначного значення, яке отримано в результаті обчислення виразу <multifield-expression>. Застосування параметра <list-variable> разом з параметром <multifield-expression> дозволяє здійснювати вибірку поля поточної ітерації, посилаючись на змінну. Крім того, шляхом додавання суфікса *-index* до імені параметра <list-variable> створюється спеціальна змінна, яка містить індекс поточної ітерації.

Функція *progn*$ повертає значення останнього виразу <expression>, обчисленого для останнього поля параметра <multifield-expression>. При порушенні синтаксису представлення функцій повертається значення FALSE.

Приклади використання обох функцій наведені нижче:

```
(progn (create$ (+ 2 4 3)  (- 8 5 7)  (/ (* 2 4 15) (+ 8 171))))
(progn (create$ (+ 2 4 3)  (- 8 5 7)  (/ (* 2 4 15) (+ 8 171)))
(printout t  ‘ – ‘  crlf))
(progn$ (create$ (+ 2 4 3)  (- 8 5 7)  (/ (* 2 4 15) (+ 8 171))  77)))
(progn$ (?v (create$ (+ 2 4 3)  (- 8 5 7)  (/ (* 2 4 15) (+ 8 171))  77))
(printout t ?v-index ‘ – ‘ ?v crlf))
```

Введення в CLIPS та їх виконанняприводить до отримання результатів, показаних на рис. 2.6

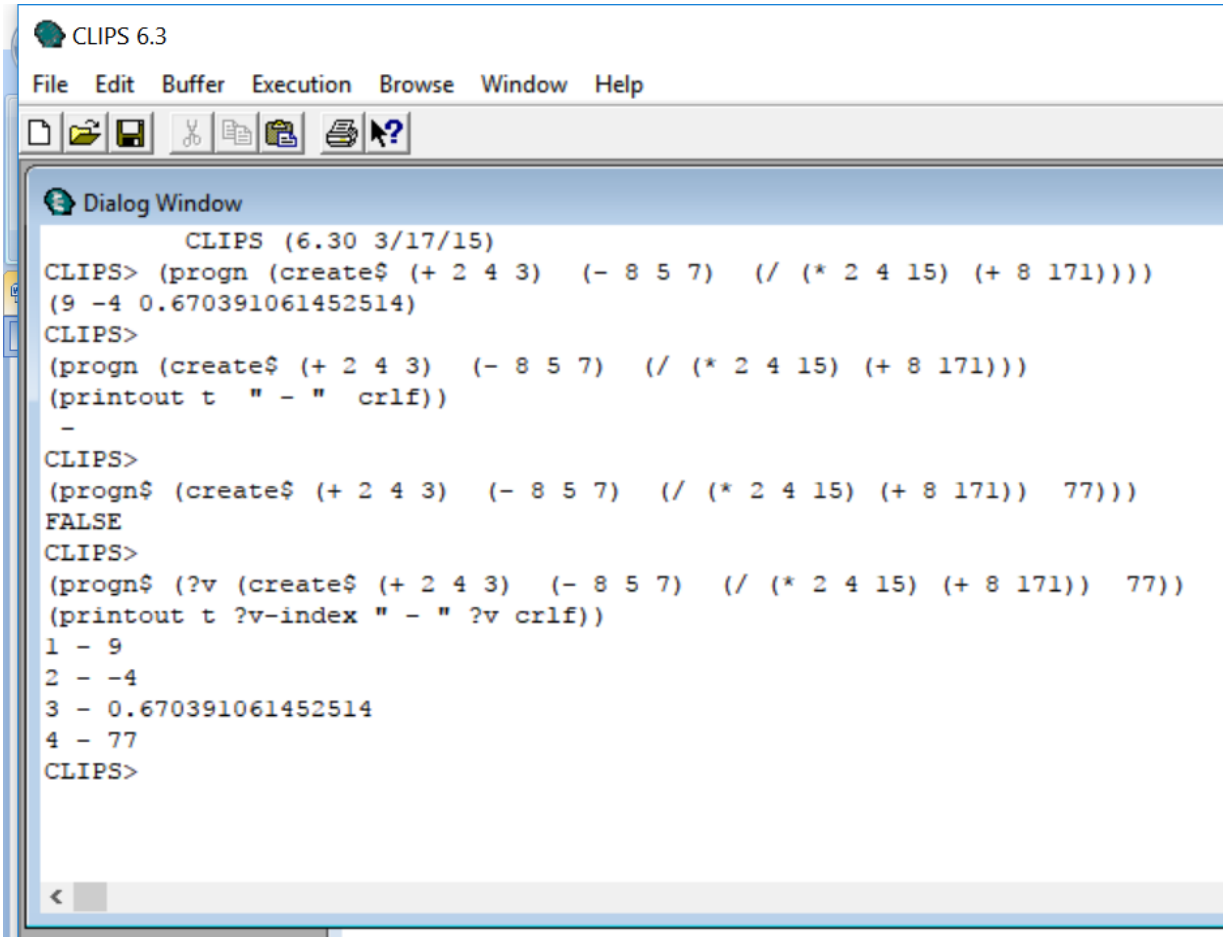

Рис. 2.6. Результати виконання CLIPS функцій progn і progn$.

Функція break.

Функція *break* (з синтаксисом (break)) перериває виконання поточних циклів функції *while*, *loop-for-count* або роботу *progn*$, а також деяких функцій, що виконують дії над набором об’єктів (зокрема, *do-for-instance*, *do-for-all-instances* і *delayed-do-for-all-instances*), в тілі яких вона безпосередньо виявляється. Зазвичай вона використовується для передчасного завершення циклу при виявленні деякої умови.

Потрібно мати на увазі, що функція *break* не повинна використовуватися в рамках функції *progn*, якщо це є неприпустимим, виходячи з зовнішнього контексту *progn*. Також, функція *break* не повинна використовуватися в якості параметра звернення до іншої функції.

Приклади використання функції *break* показані на рис. 2.7.

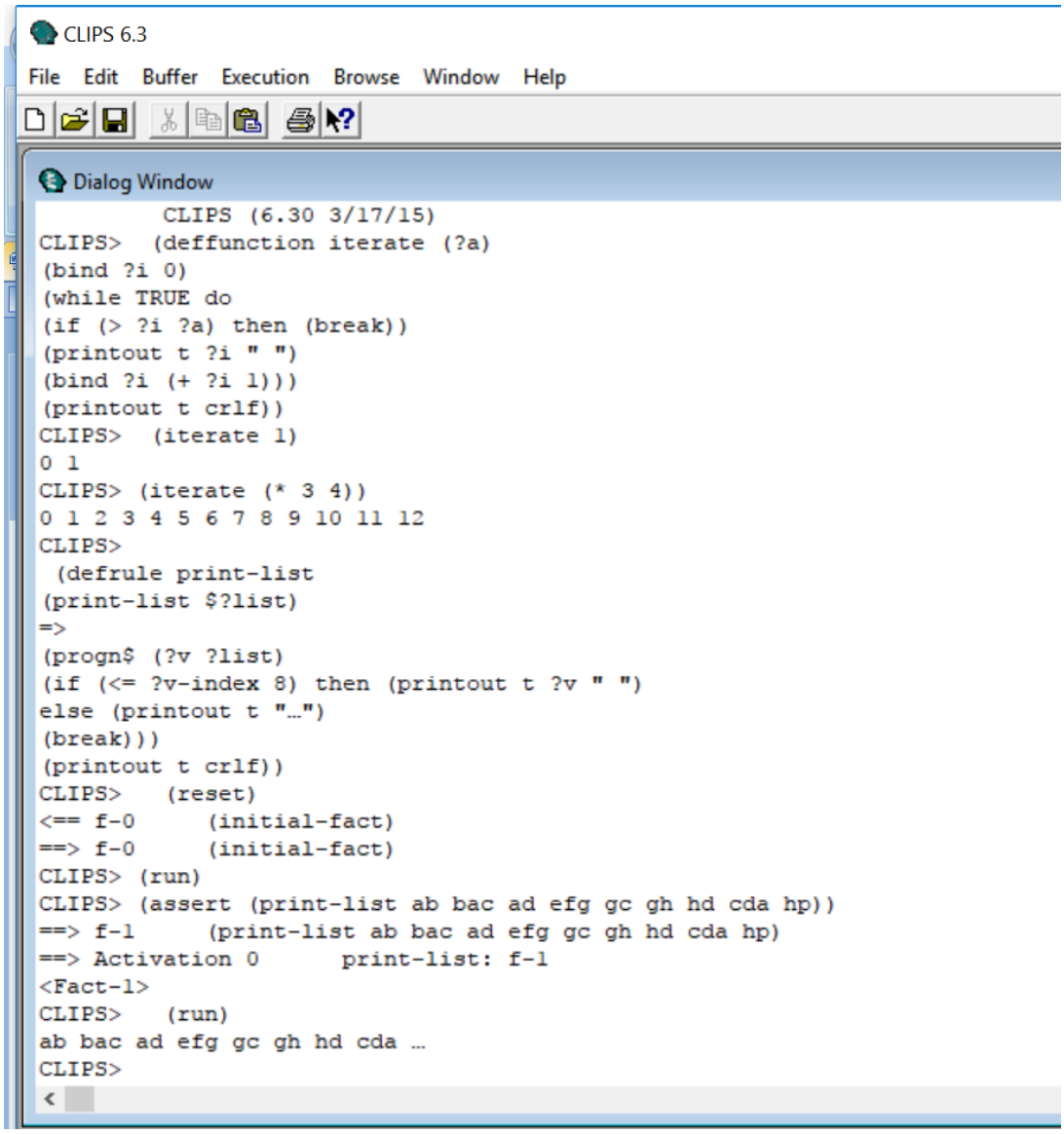

Рис. 2.7. Приклади використання функції *break.*

Функція return.

Функція *return* не тільки дозволяє екстрено зупинити виконання циклу, припинити виконання правої частини правила, але і надає можливість припинити роботу конструкції *deffunction*, що виконується в даний момент, а також родового методу функції обробника повідомлень.

Функція *return*, застосовувана разом з конструкцією *deffunction*, має наступний синтаксис:

(return [<expression>])

Якщо в цьому визначенні задано вираз <expression>, то результат обчислення цього виразу повертається як значення конструкції *deffunction*. Якщо параметр з виразом <expression> відсутній, то відсутнє і значення, що повертається.

Функція halt.

Функція *halt* зазвичай використовується у правій частині правила для зупинки виконання правил, що знаходяться в робочому списку. Після її виклику (функція не вимагає параметрів; її синтаксис – (halt)) припиняється виконання будь-яких дій, заданих у правій частині запущеного правила, а управління передається до командного рядка CLIPS.

Виконання команди не впливає на робочий список правил – в ньому продовжують перебувати всі правила, активовані до моменту виклику функції *halt*.

Функції порівняння за типом і значенням eq і neq.

Синтаксис цих функцій має такий вигляд:

(eq <expression><expression> +)
(neq <expression><expression> +)

Функція *eq* повертає значення TRUE, якщо її перший аргумент має той же тип і значення, що і всі наступні аргументи (якщо вони присутні). В іншому випадку функція повертає значення FALSE.

Функція *neq*, навпаки, повертає значення TRUE, якщо її перший аргумент має не той же тип і значення, що і всі наступні аргументи, і значення FALSE – в іншому випадку.

Важливою особливістю функцій є те, що вони порівнюють як значення аргументів, так і їх типи. Наприклад, результатом виконання виразу (eq 3 3.0) буде значення FALSE (і, навпаки, результатом виконання виразу (neq 3 3.0) буде зна-

чення TRUE), оскільки число 3 належить типу *integer*, а число 3.0 – типу *float*:

```
CLIPS> (eq 3 3.0)
FALSE
CLIPS> (neq 3 3.0)
TRUE
CLIPS>
```

Порядок виконання роботи.

1. Ознайомитися з особливостями виклику функцій у CLIPS та з особливостями застосуванням основних математичних функцій CLIPS. Освоїти особливості роботи в CLIPS з використанням інтерфейсу.
2. Завантажити середовище CLIPS 6.30 для ОС Windows запуском файлу CLIPSIDE64.exe чи CLIPSIDE32.exe (версія 6.30 3/17/15) – в залежності від розрядності версії вашої ОС.
3. З використанням командного рядка здійснити програмування математичних виразів та розрахувати їх значення:

$$\sqrt{5^4 + \sqrt{7^2 + 1} + \ln 20.5};$$

$$\sin 1 + 1/(\cos 1 - 2);$$

$$2e^4 - 4 - |\sin 6^2|;$$

$$3^3 - e^{7+\sin 3};$$

$$|3e^3 - 2\ln 34|,$$

а також інших, запропонованих індивідуально для кожного студента. Список пропонованих для розрахунку індивідуальних завдань представлений у матеріалах до даної роботи (див. Додаток 1). Вибір відповідного завдання окремим студентом – за порядковим номером його прізвища в списку студентів групи.

4. Провести ці ж обчислення, використовуючи для запису коду та його вве-

дення вбудований текстовий редактор CLIPS, для чого:

- увійти в меню Fail і натиснути пункт New;
- написати код для програмування виразу, що розраховується;
- ввести у вікні Untitled написаний код для розрахунку потрібного математичного виразу;
- замаркувати введені команди та ввести в CLIPS – натисканням комбінації клавіш CTRL+M або ж натиснувши кнопку Batch Selection з меню Buffer;
- при правильному синтаксисі введених команд (у командному рядку не з'явиться попередження про помилку), буде проведений розрахунок і виведений результат в основному вікні;
- якщо після введення в командному рядку з'явиться повідомлення про помилку, перевірте правильність синтаксису написання коду і після виправлення помилок введіть його повторно. Правильність синтаксису написаного коду буде підтверджена отриманим результатом розрахунку.

5. Створити нові внутрішні функції CLIPS (наприклад, функції розрахунку площі, об'ємів геометричної фігури або ін., результат яких враховує введення значень кількох параметрів) і провести розрахунки з їх використанням.
6. Написати звіт про виконану роботу; при оформленні звіту використовувати скріншоти окремих етапів роботи.

Контрольні запитання:

1. Який синтаксис визначень застосовується в CLIPS? Які його особливості?
2. Яка нотація вважається базовою при представленні конструкцій і команд в CLIPS?
3. Що таке термінальні і нетермінальні символи CLIPS і яку роль вони відіграють?
4. Яку роль відіграють при написанні коду в CLIPS різні символи (круглі та ква-

дратні дужки, лапки, апострофи)?

5. Що таке процедурна парадигма програмування? Для чого вона застосовується у CLIPS?
6. Що розуміється під командами і функціями в CLIPS і яка різниця між ними?
7. Які команди головного меню CLIPS можна вважати основними?
8. Яка форма представлення функцій у CLIPS?
9. В чому полягають особливості синтаксису написання програми виконання математичних операцій у CLIPS?
10. Яку роль відіграють дужки в синтаксисі написання функцій і команд CLIPS?
11. Які команди найчастіше використовуються при роботі в CLIPS?
12. Що таке вбудовані математичні функції CLIPS і в чому полягає особливість їх використання?
13. Що потрібно враховувати при програмуванні виконання операцій з математичними функціями в CLIPS?
14. На що потрібно звернути увагу насамперед, якщо реакцією середовища на введений функціональний вираз буде FALSE?
15. Який порядок запуску введеної команди (програми) в CLIPS? Які є варіанти запуску на виконання команд у CLIPS?
16. Які переваги використання спеціалізованого редактора програміста (наприклад, Notepad++ або іншого) при програмуванні математичних виразів?
17. Який порядок операцій потрібно зберігати при програмуванні складних математичних виразів?
18. Що таке локальні та глобальні змінні CLIPS та у чому полягає різниця між ними?
19. Як задаються глобальні змінні в CLIPS?
20. Які маніпуляції і з допомогою яких команд можна проводити з глобальними змінними?
21. Як розрізняються позначення локальних та глобальних змінних?
22. Яким чином привласнюються (змінюються) значення локальних чи глобаль-

них змінних?

23. Виконання яких функцій у правилах CLIPS головним чином забезпечують процедурні функції?
24. Що таке внутрішні і зовнішні функції CLIPS?
25. Які операції забезпечує використання внутрішніх функцій CLIPS?
26. Чому і коли виникає потреба у створенні нових функцій?
27. Яким чином можна створити нові функції у середовищі CLIPS?
28. Який синтаксис конструктора *deffunction*?
29. Де можуть бути використані функції і математичні вирази при створенні бази знань експертної системи?

Література:

1. CLIPS Reference Manual CLIPS Basic Programming Guide і CLIPS Basic Programming Guide Version 6.31 June 12th 2019. – Доступне на: http://www.clipsrules.net/bpg631.pdf
2. Джарратано Джозеф. Экспертные системы. Принципы разработки и программирование, 4е изд: Пер. с англ. / Джозеф Джарратано, Гари Райли. – М., Изд. дом “Вильямс’, 2007. – 1152 с.
3. Частиков А. П. Разработка экспертных систем. Среда CLIPS. / А. П. Частиков., Т. А. Гаврилова, Д. Л. Белов. – СПб., «БХВ-Петербург», 2003. – 608 с.

Лабораторна робота 3.

Тема: РОБОТА З ФАКТАМИ В CLIPS.

Мета роботи: Отримати навички введення в CLIPS впорядкованих та невпорядкованих фактів та їх перетворень.

Завдання до роботи:

Вивчити роль та значення фактів у експертних системах. Засвоїти особливості різних методів представлення і введення впорядкованих і невпорядкованих фактів у середовищі CLSPS та дії функцій, призначених для маніпуляції з введеними фактами – перевірки наявності введених фактів у робочій пам'яті, їх модифікації і зміни, видалення, очищення списку фактів, глибокого очищення середовища. Засвоїти особливості введення одиночних фактів та їх масивів і команд, які використовуються при цьому.

Вихідні матеріали:

3. ФАКТИ ТА ЇХ РОЛЬ В CLIPS.

Робота експертної системи, за визначенням, ґрунтується на використанні закладених в неї знань, і тому її функціонування неможливе без наявності в ній спеціально створеної бази знань. Найчастіше ці знання в експертних системах представляються набором певних фактів і правил, які задаються з використанням деякої мови опису знань – такі системи ще називаються продукційними. Тому будь-яка експертна система повинна володіти арсеналом засобів для створення, введення фактів і правил, а також маніпуляцій ними.

CLIPS, як спеціалізоване середовище, призначене для створення експертних

систем продукційного типу, не є винятком з цього правила і надає широкі можливості як для створення, накопичення, зберігання й обробки фактів і правил, так і їх для використання з метою отримання потрібного логічного висновку.

Загалом, CLIPS підтримує дві парадигми подання знань – евристичну й процедурну. У евристичній парадигмі для подання знань використовуються факти і правила, представлені у декларативному вигляді, а у процедурній – використовуються механізми створення і застосування глобальних змінних, функцій, а також т. з. *родових функцій*.

У CLIPS застосовується два типи представлення фактів – у вигляді *впорядкованих* (ordered facts) і *невпорядкованих* (non-ordered facts) *фактів,* або *шаблонів* (template facts). Посилатися на дані, що зафіксовані у факті, можна або використовуючи строго задану позицію значення в списку даних для впорядкованих фактів, або ж вказуючи ім'я для шаблонів і значення слотів у невпорядкованих фактах.

3.1. Факти в CLIPS.

Для успішного рішення експертною системою, створеною в середовищі CLIPS, поставленої задачі їй повинна бути надана інформація, на підставі якої вона могла б здійснювати свої міркування і формувати висновок. Факти і є тими «фрагментами» інформації, які використовуються в мові CLIPS.

Факти є однією з основних форм представлення інформації в системі CLIPS, складаються з символьного поля імені-відношення, за яким слідує або нелімітована кількість символьних полів (розділених недрукованими символами, зазвичай – пробілами) (у випадку впорядкованих фактів), або слотів і пов'язаних з ними значень (у невпорядкованих фактах), та зберігаються в оперативній пам'яті комп'ютера. Весь факт (як і кожен слот) обмежується відкриваючою і закриваючою круглими дужками.

Таким чином, кожен факт представляє фрагмент інформації, який поміщається в поточний список фактів (*fact-list*) пам'яті ЕС і є основною одиницею да-

них, що використовуються для активування правил. На кількість фактів в списку і на об’єм інформації, який може бути збережений у кожному факті, загалом ніяких обмежень немає; обмеження може бути зумовлене тільки розміром пам’яті комп’ютера.

Факт у CLIPS може описуватися або індексом, або ж його адресою (змінним покажчиком, що зберігає індекс факту). Кожного разу, коли факт додається (змінюється), йому привласнюється унікальний цілочисельний індекс. Індекси фактів починаються з нуля і для кожного нового або зміненого факту збільшуються на одиницю. Ідентифікатор факту записується у вигляді f-<індекс>. Наприклад, запис f-10 служить для позначення факту з індексом 10. Кожного разу після виконання очищення пам’яті (тобто виконання команд *reset* чи *clear*) виділення індексів фактів починається з нуля.

Для переглядання поточного списку фактів використовується команда *facts*.

Як уже згадувалося, CLIPS підтримує два формати представлення фактів: врегульовані (або впорядковані) і неврегульовані (або невпорядковані, чи шаблонні) факти. Практично, будь-який фрагмент інформації можна представити у вигляді як впорядкованого, так і невпорядкованого факту. Наприклад, той факт, що студент Іван Іванович Іванів, віком 21 рік, має голубі очі і чорне волосся, може бути представлений у вигляді як впорядкованого факту:

(student Ivan Ivanovych Ivaniv aged 21 has blue eyes and black hair),

так і невпорядкованого:

```
(student
(name  «Ivan  Ivanovych  Ivaniv»)
(age  21)
(eye-color blue)
(hair-color black)).
```

Вираз символьного типу student тут є ім’я-відношення для даного неупорядкованого факту, а сам факт містить чотири слоти: name (ім’я), age (вік), eye-color (колір очей) і hair-color (колір волосся). Значенням слота name є «Ivan Ivanovych Ivaniv», значенням слота age – число 21, значенням слота eye-color – blue, а значенням слота hair-color – black. Слід зазначити, що порядок, в якому задані слоти у невпорядкованому факті, не має значення для відображення сенсу інформації, закладеної в ньому.

3.1.1. Впорядковані факти.

Впорядковані факти складаються з виразу символьного типу, за яким слідує необмежена послідовність (можливо, і порожня) з полів, розділених недрукованими символами (пропусками). Перше поле запису визначає «ім’я відношення» (або «зв’язок» факту), яке застосовується до полів, що залишилися. Весь запис поміщається в круглі дужки.

За винятком першого поля, що обов’язково повинне бути типу *symbol*, інші поля у факті можуть зберігати дані будь-якого примітивного типу CLIPS. Потрібно також пам’ятати, що слова: test, and, or, not, declare, logical, object, exist й forall – зарезервовані й не можуть бути використані як перше поле впорядкованого факту. Синтаксис впорядкованого факту:

(дані_типу_symbol [поле]*).

Приклади впорядкованих фактів:

(student Sergiy Petrenko)
(classmates Ivaniv Petriv Sydorenko)
(color red).

Термін «зв'язок» означає, що даний факт належить деякому певному конструктору або неявно оголошеному шаблону. Тому що, насправді, при виявленні інтерпретатором CLIPS кожного впорядкованого факту ним автоматично створюється неявна конструкція *deftemplate* (див. 1.2) для цього факту (на відміну від явної конструкції *deftemplate*, що створюється із застосуванням визначення *deftemplate* і задає опис невпорядкованого факту). Виходячи з цього, можна сказати, що впорядкований факт – це, по суті, один багатозначний слот, який використовується для зберігання всіх значень, що слідують за його іменем.

3.1.2. Невпорядковані факти.

На відміну від впорядкованих, невпорядковані (шаблонні) факти дають можливість користувачеві задавати певну абстрактну структуру факту, задаючи імена кожному з полів (слотів) факту з певним іменем-відношенням. Це означає, що для забезпечення можливості створення конкретного факту інтерпретатору CLIPS необхідно попередньо повідомити інформацію про те, яким є список допустимих слотів для факту з вказаним іменем, тобто створити шаблон факту. Причому, як уже згадувалося, порядок слотів *у* неупорядкованому факті не важливий. Визначення невпорядкованого факту, як і впорядкованого, обмежується круглими дужками.

Конструктор *deftemplate.*

Для задання шаблону, який потім може використовуватися і при доступі до полів (слотів) невпорядкованого факту за їх іменами, використовується конструктор *deftemplate.* Його створення і введення в CLIPS приводить до появи в поточній базі знань системи інформації про шаблон факту, за допомогою якого в неї надалі можна буде додавати факти, що відповідають заданому шаблону.

Синтаксис конструктора *deftemplate*:

(deftemplate <relation-name>[*“Необов’язковий коментар”*]

<slot-definition>

...

<slot-definition>)

Імена шаблонів і слотів повинні бути значеннями типу *symbol*; на імена шаблонів, як і у випадку впорядкованих фактів, також поширюється заборона на використання слів, зарезервованих системою.

Коментарі, зазвичай, описують призначення шаблона; щоб вони зберігалися в базі знань системи і були доступні при перегляді визначення шаблона, їх потрібно помістити в лапки. Крім даного типу коментарів, у конструкторі *deftemplate* також застосовні звичайні коментарі CLIPS, що починаються із символу ; – такі коментарі повністю ігноруються системою CLIPS.

Опис синтаксису <slot-definition> визначається таким чином:

<slot-definition> ::=(slot <slot-name>) | (multislot <slot-name>).

Відповідно, слот може бути простим або складеним (мультислотом).

Прості слоти факту, задані за допомогою ключового слова *slot* у відповідних конструкціях *deftemplate*, містять тільки одне значення примітивного типу CLIPS (такі слоти ще називають *однозначними слотами).* У простому слоті може бути збережене лише одне значення.

Складені слоти факту, задані за допомогою ключового слова *multislot*, можуть містити будь-яку кількість (від нуля і більше) значень декількох примітивних типів CLIPS – такі слоти ще називають *багатозначними слотами;* у складеному слоті факту може зберігатися список.

Отже, <slot-definition>в простому випадку складається з:

1. ключового слова *slot* або *multislot*, що визначає тип слота;

2. імені слота, яке є значенням типу *symbol.*

3. необов'язкового обмеження на тип значення, що зберігається в слоті.

При створенні шаблона за допомогою конструктора *deftemplate* кожному полю можна призначати певні атрибути, що задають або значення за замовчуванням (якщо при створенні факту не задане конкретне значення слота), або обмеження на значення слота.

Значення будь-якого слота за замовчуванням у шаблоні можна задати за допомогою атрибуту *default* (визначає статичне значення за замовчуванням, що обчислюється один раз при конструюванні шаблона і зберігається разом із шаблоном) чи *default-dynamic* (цей атрибут щоразу при додаванні факту за даним шаблоном визначає вираз, що привласнюється відповідному слоту).

Значення атрибуту *default* може задаватися за допомогою службових слів ?DERIVE (тоді значення атрибуту буде витягатися з обмежень, заданих для даного слоту) або ?NONE (у цьому випадку значення поля обов'язково повинне бути явно задане в момент виконання операції додавання факту). За замовчуванням, для всіх слотів установлюється атрибут (default ?DERIVE).

Для перегляду наявних у поточній базі знань шаблонів можна скористатися командою *get-deftemplate-list* (виводить список наявних в пам'яті шаблонів у діалоговому вікні) або менеджером шаблонів Deftemplate Manager (виводить список наявних в пам'яті шаблонів у окремому вікні; для його виведення потрібно вибрати пункт Deftemplate Manager в меню Browse); вигляд вікна CLIPS з результатами виконання команди (get-deftemplate-list) та менеджера шаблонів представлений на рис. 3.1.

Менеджер шаблонів дозволяє в окремому вікні переглядати список всіх шаблонів – зокрема, показує кількість шаблонів, доступних у поточній базі знань (на рис. 1 Deftemplate Manager – 2 Items), видаляти обраний шаблон (кнопка Remove) і переглядати всі його властивості (імена й типи слотів) у діалоговому вікні (кнопка Pprint).

Повідомлення про додавання й видалення невпорядкованих фактів, що створюються з використанням даного шаблона, також можна спостерігати у діа-

логовому вікні CLIPS. Для цього потрібно відмітити прапорцем пункт Fakts у вікні Watch Options меню Execution.

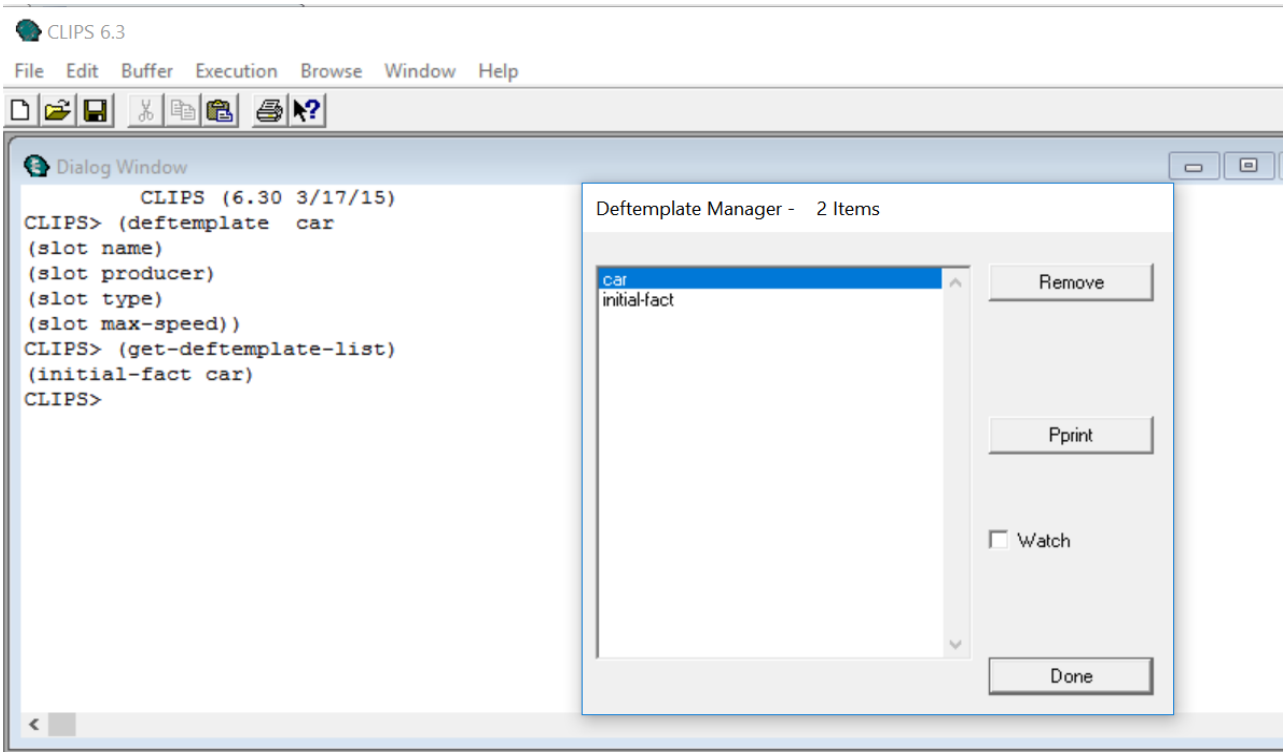

Рис. 1. Вигляд менеджера шаблонів у CLIPS.

У випадку, якщо при додаванні нового шаблона за допомогою конструктора *deftemplate* відбулася помилка, користувач одержить відповідне повідомлення.

Перевизначення вже існуючого шаблона приводить до виключення попереднього визначення. Шаблон не може бути перевизначений доти, поки він використовується (наприклад, фактом або правилом).

Додавання невпорядкованих фактів.

Після визначення відповідної конструкції *deftemplate* невпорядковані факти можна додавати у пам’ять системи за допомогою команди *assert* (цією командою факти додаються поодинці) або з використанням конструктиву *deffacts* (ним додається масив невпорядкованих фактів). Особливості застосування цих команд бу-

дуть розглянуті нижче.

Розглянемо приклад: Визначимо шаблон для введення невпорядкованого факту student і введемо його:

```
(deftemplate student
(slot name)
(slot age)
(slot eye-color)
(slot hair-color)
)

(assert (student
(name «Іван Іванович Іваненко»)
(age 23)
(eye-color blue)
(hair-color black)))
```

В результаті послідовного введення в діалоговому вікні визначення шаблона для невпорядкованого факту student та самого факту в пам’ять CLIPS вноситься відповідний факт, про що сповіщає повідомлення в діалоговому вікні:

```
==> f-1     (student (name «Іван Іванович Іваненко»)
 (age 23)
 (eye-color blue) (hair-color black))
<Fact-1>.
```

Результати виконання введення в CLIPS відповідного шаблона та невпорядкованих фактів показані на рис рис. 3.2.

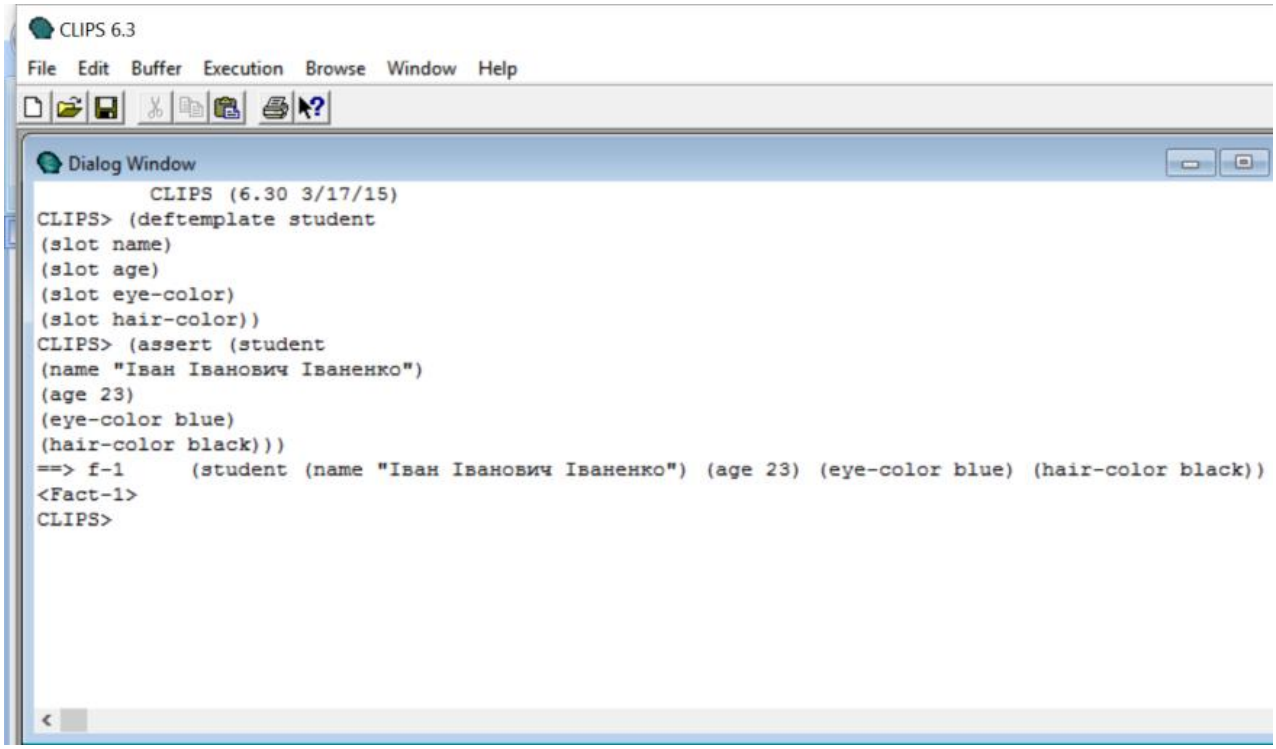

Рис. 3.2. Результати введення в CLIPS невпорядкованого факту student.

3.2. Дії з фактами.

Уся наявна в CLIPS фактична інформація групується і зберігається у списку фактів.

CLIPS відрізняє невпорядковані факти від впорядкованих за першим полем факту, яке, нагадаємо, для фактів обох типів повинно бути значенням типу *symbol*. Якщо це значення відповідає імені деякого шаблона, тоді факт – упорядкований.

З фактами можна здійснювати різноманітні дії: додавати і видаляти факти, які представляють певну інформацію, а також здійснювати певні маніпуляції з ними – змінювати, дублювати, тощо – вводячи відповідні команди або безпосередньо з клавіатури, або із програми.

Для того, щоб мати можливість спостерігати за процесом додавання, видалення або зміни фактів безпосередньо у діалоговому вікні, необхідно викликати підменю Watch меню Execution і встановити прапорець напроти пункту «Facts» в

списку параметрів трасування (або ж використати комбінацію клавіш Ctrl+W) (рис. 3.3).

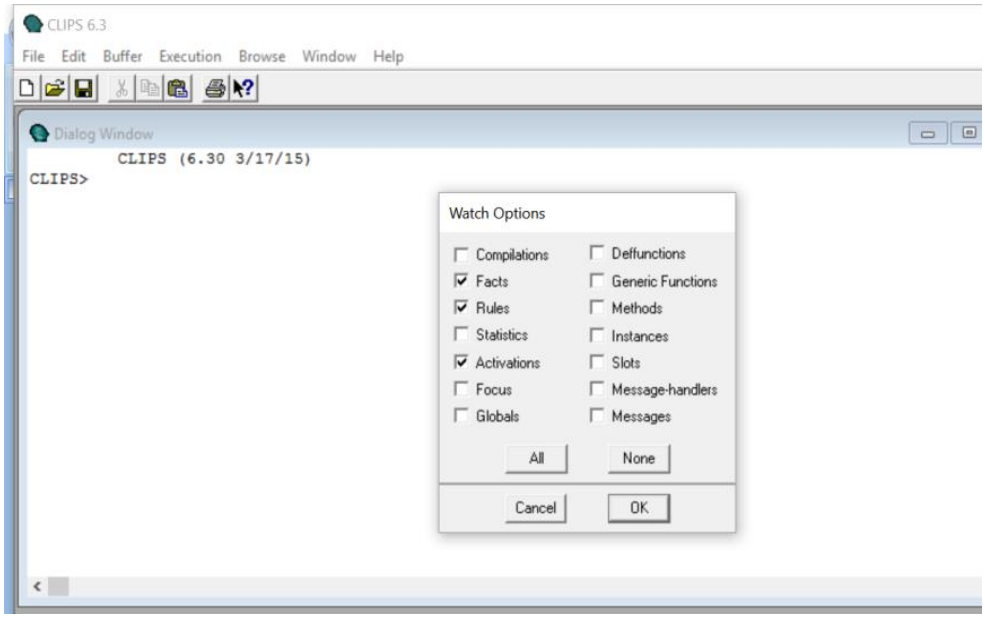

Рис. 3.3. Установка параметрів трасування.

Для здійснення маніпуляцій з фактами в CLIPS використовується низка функцій (команд). Розглянемо особливості застосування деяких з них.

Команда *facts*.

Перевірити поточний стан списку фактів можна або за допомогою введення в діалоговому вікні команди *facts* (тоді список усіх наявних в системі фактів буде виведений в діалоговому вікні), або викликавши пункт 1 Facts Window меню Window (тоді список наявних в системі фактів буде виведений у вікні Facts (MAIN), яке з'являється в полі дисплея; рис. 3.4).

Якщо кількість фактів, що містяться в списку, є великою, то іноді виникає необхідність мати можливість розглядати тільки потрібну на даний момент його частину. Така можливість забезпечується шляхом задання додаткових параметрів команди *facts*. Загалом, повний синтаксис команди *facts* такий:

(facts [<start> [<end> [<maximum>]]])

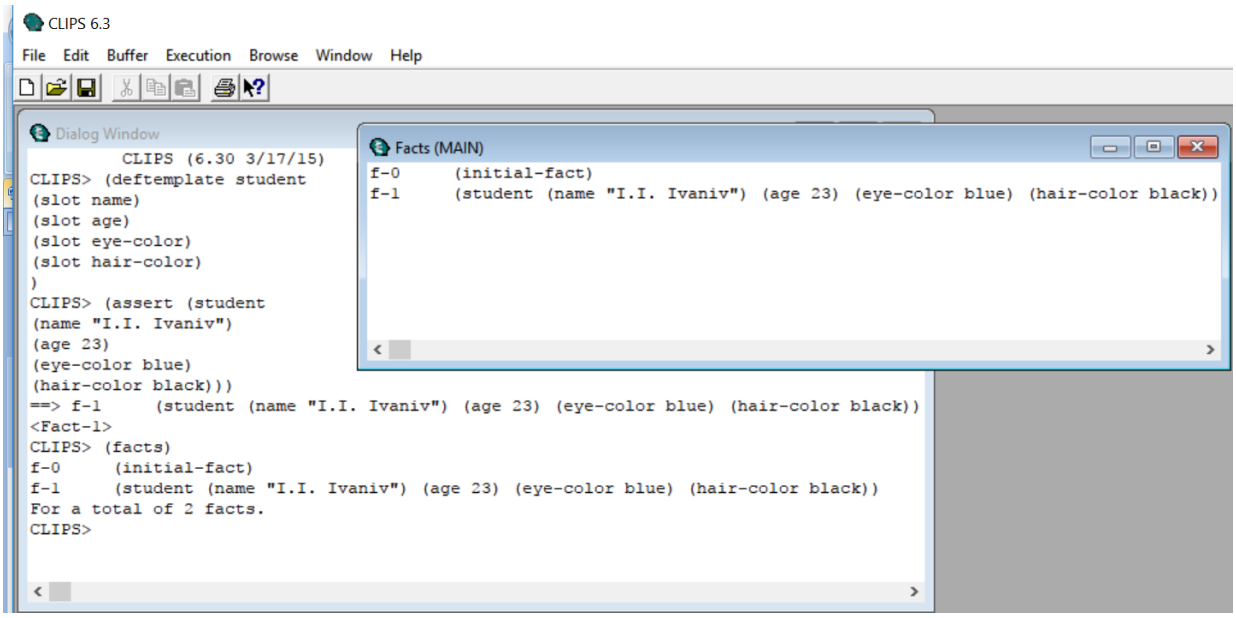

Рис. 3.4. Вигляд діалогового вікна з поточним списком фактів після введення команди (facts) та вікна із списком фактів Facts (MAIN).

У цій команді як параметри <start>, <end> і <maximum> застосовуються позитивні цілі числа, які визначають початкове, кінцеве та максимальне значення індексів фактів, які наявні в пам’яті системи і які потрібно переглянути.

Функція *assert.*

Нові факти можуть бути додані до наявних у списку з використанням команди *assert*, яка має такий синтаксис:

(assert <fact>+)

Параметрами функції є послідовність фактів, які підлягають додаванню в список фактів; кількість фактів, що додаються, довільна.

Розглянемо приклад використання функції *assert* для введення впорядкованого факту (приклад введення невпорядкованого факту показано в п. 1.2). Для цього перейдемо в діалогове вікно CLIPS і введемо:

(assert (weather is fine))

При введенні команд необхідно чітко слідувати їх синтаксису, зокрема – бути уважними з дужками. У тому випадку, якщо команда введена правильно, у список фактів додасться факт (weather is fine) і діалоговому вікні CLIPS з'явиться результат її виконання:

```
==> f-1 (weather is fine)
<Fact-1> ,
```

що означає, що факт доданий в систему і отримав номер 1. При успішному виконанні функція *assert* повертає адресу останнього доданого факту.

При додаванні факту можна використовувати і виклики функцій; наприклад, виконання команди введення факту

(assert (totalcost (* 10 13)))

викличе функцію множення і її результат запишеться як поле факту. При цьому в пам'ять додасться факт (totalcost 130) і в діалоговому вікні з'явиться повідомлення:

```
CLIPS> (assert (totalcost (*  10 13)))
==> f-1     (totalcost 130)
<Fact-1>
```

Якщо при додаванні деякого факту була допущена помилка (напр., повторне введення вже існуючого факту), команда припиняє роботу і повертає значення FALSE; якщо при цьому має місце синтаксична помилка, в діалоговому вікні виводиться відповідне повідомлення. Результати неправильного введення фактів відображаються у діалоговому вікні CLIPS і для випадку вказаного вище повторного введення факту мають вигляд:

```
CLIPS> (assert (totalcost (*  10 13)))
==> f-1     (totalcost 130)
<Fact-1>
CLIPS> (assert (totalcost (*  10 13)))
FALSE
CLIPS
```

Функція *assert* – одна з команд, які найчастіше застосовуються у системі CLIPS. Без її використання неможливо написати код навіть найпростішої експертної системи та запустити її на виконання в середовищі CLIPS.

Функція *assert-string*.

Функція *assert-string* з синтаксисом

(assert-string <string-expression>)

перетворює символьний рядок (у тому вигляді, у якому він набирається; символьний рядок при наборі повинен бути поміщений у лапки, а сам факт обмежений круглими дужками) у факт, додає його до бази знань і повертає індекс знову доданого факту.

Функція перетворює заданий строковий вираз у факт CLIPS, розділяючи окремі слова на поля, з врахуванням наявних у системі на даний момент шаблонів і може працювати як з упорядкованими, так і з невпорядкованими фактами. Одним викликом функції *assert-string* можна додати тільки один факт.

Якщо додавання факту пройшло вдало, функція повертає індекс тільки що доданого факту; у зворотному випадку функція повертає повідомлення про помилку й значення FALSE. Функція *assert-string* не дозволяє додавати факт у випадку, якщо такий факт уже присутній *у* базі знань.

Приклад: Виконання CLIPS команди введення символьного факту

(assert-string «(book-name \»CLIPS User Guide\»)»)

приведе до додавання в список фактів факту з індексом f-1:

```
CLIPS> (assert-string «(book-name \»CLIPS User Guide\»)»)
==>
 f-1     (book-name «CLIPS User Guide»)
<Fact-1>
CLIPS>
```

Потрібно мати на увазі, що при необхідності запис в символьному факті додаткових лапок (або косих рисок) здійснюється за допомогою використання зворотних рис (backslash) або їх відповідних комбінацій. Напр., при введенні комбінації \» у введеному факті буде відображено подвійні лапки.

Команда *retract.*

Для видалення фактів з поточного списку фактів в CLIPS передбачена функція *retract*, яка видаляє з поточного списку довільну кількість фактів, і має такий синтаксис:

(retract <fact-index>+ | *)

Як аргументи команди *retract* повинні бути вказані індекси фактів, що відносяться до одного або декількох фактів.

Кожним викликом цієї функції можна видалити довільне число фактів. У випадку, якщо включений режим перегляду зміни списку фактів (див. рис. 3), то відповідне інформаційне повідомлення відображатиметься у діалоговому вікні CLIPS при видаленні кожного факту.

Аргумент <fact-index>може бути або змінною, пов’язаною з адресою факту (адреса факту повертається командою *assert* чи з використанням певного правила), або індексом факту без префікса (напр., 5 для факту з індексом f-5), або виразом, що обчислює цей індекс (напр., (+ 1 3) для факту з індексом f-4). Якщо як аргумент функції *retract* використовувався символ *, то тоді командою (retract *) з

поточного списку будуть видалені всі факти. Функція *retract* не має значення, що повертається.

Розглянемо роботу команди *retract* на прикладі. Перейдемо в діалогове вікно CLIPS і послідовно введемо команди:

(clear)

(assert (a) (b) (c) (d) (e) (f))

(retract 0 (+ 0 2) (* 2 2))

Результатом виконання приведеної вище послідовності команд буде:

1. Очищення списку фактів.
2. Додавання в систему шести фактів.
3. Видалення з системи фактів з номерами 0, 2 і 4.

Потрібно мати на увазі, що список фактів у вікні Facts кожною виконаною командою змінюється. При успішному виконанні вказаних вище команд в списку залишаться факти (a), (c), (e) і (f).

Результати послідовного виконання CLIPS вказаних у вище розглянутому прикладі списку команд показані на рис. 3.5.

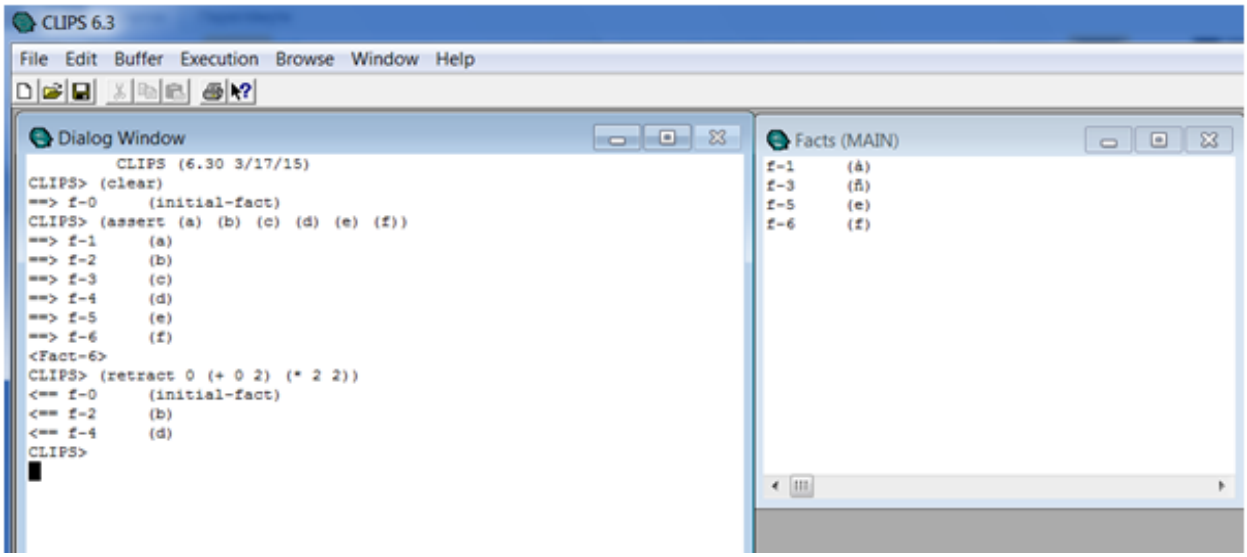

Рис. 3.5. Результати введення фактів в CLIPS та видалення деяких з них.

Функція *fact-relation.*

Функція *fact-relation* з синтаксисом

(fact-relation <fact-index>)

дозволяє визначити відношення (зв'язок) факту з вказаним індексом (або адресою) із шаблоном, заданим явно чи неявно, за допомогою якого створений факт. Цей зв'язок визначається за першим полем факту, яке завжди є простим полем і використовується CLIPS як ім'я шаблона, з яким зв'язаний факт. Функція повертає значення першого поля факту (ім'я факту), якщо даний факт існує, або значення FALSE – якщо не існує.

Функція *fact-existp.*

Для визначення, чи міститься в поточному списку фактів факт з вказаним індексом, використовується функція *fact-existp* з синтаксисом:

(fact-existp <fact-index>).

У випадку, якщо такий факт присутній у поточному списку фактів, то функція fact-existp повертає значення TRUE, якщо ні – FALSE.

Наприклад, послідовне виконання CLIPS команд:

```
(clear)
(assert-string «(Weather is fine)»)
(fact-existp 1)
(retract 1)
(fact-existp 1)
```

приведе до такого результату:

```
CLIPS> (assert-string «(Weather is fine)»)
```

```
==> f-1     (Weather is fine)
<Fact-1>
CLIPS> (fact-existp 1)
TRUE
CLIPS> (retract 1)
<== f-1     (Weather is fine)
CLIPS> (fact-existp 1)
FALSE
CLIPS>
```

Команда *save-facts.*

Команда збереження списку фактів у файл *save-facts* має синтаксис:

(save-facts <file-name> [visible | local <deftemplate-names>*])

і зберігає факти з поточного списку фактів у текстовий файл з вказаним іменем.

У функції існує можливість обмежити область фактів, що зберігаються. Для цього служить аргумент <межі-видимості>, який може приймати значення local або visible. У випадку, якщо цей аргумент приймає значення visible, то зберігаються всі факти, які є у системі на момент запису; якщо ж аргумент – ключове слово local, то зберігаються тільки факти з поточного модуля. За замовчуванням аргумент <межі-видимості> приймає значення local. Після аргументу <межі-видимості> може бути представлений список визначених у системі шаблонів. У цьому випадку будуть збережені тільки ті факти, які створені із використанням зазначених шаблонів.

У випадку успішного виконання, команда повертає значення TRUE, а у випадку невдачі – відповідне повідомлення про помилку. Якщо зазначений файл уже існує, він буде перезаписаний.

Приклад використання функції *save-facts*. Введемо з командного рядка команду

(save-facts file-l local students)

У разі успішного виконання команди у поточний у текстовий файл з іменем file-1 будуть збережені всі наявні факти поточного модуля, пов’язані з шаблоном students.

Команда *load-facts.*

Для завантаження збережених раніше файлів використається команда *load-facts* з синтаксисом:

(load-facts <file-name>)

яка додає в поточний модуль факти, що записані у збереженому раніше файлі <file-name>. У разі успіху повертає символ TRUE; в іншому – FALSE і відповідне повідомлення про помилку.

При завантаженні фактів командою *load-facts* необхідно пам’ятати, що якщо у записаному файлі є факти, пов’язані з явно створеними за допомогою конструктора *deftemplate* шаблонами, то на момент завантаження всі необхідні шаблони повинні бути вже визначені в системі. Якщо ця умова не буде виконана, то завантаження фактів закінчиться невдачею.

Вказані вище функції і команди загалом є універсальними, тобто придатні для маніпуляцій з будь-якими фактами CLIPS – не важливо, чи вони впорядковані, чи ні. Використання деяких інших функцій стосується лише невпорядкованих фактів і буде розглянуто нижче.

Повний список функцій CLIPS та їх призначення можна подивитися у документації на CLIPS, чи в [1].

3.3. Конструктор *deffacts*.

Факти можна включати в базу даних не поодинці, а цілим масивом. Таке введення зручне, наприклад, при введенні фактів, що представляють початкові знання, або множини фактів, введення яких вимагає набору однотипних команд. Для цього в CLIPS призначений конструктор *deffacts,* синтаксис якого такий:

(deffacts <deffacts name> [<optional comment>] <facts>*)

Услід за ключовим словом deffacts знаходиться обов'язкове ім'я цієї конструкції. Як ім'я може використовуватися будь-який допустимий вираз типу *symbol*, який служить більш як інформативний атрибут, ніж як ідентифікатор. За ім'ям слідує необов'язковий коментар, який зберігається у визначенні конструкції *deffacts* після завантаження відповідного визначення системою CLIPS. Услід за ім'ям або коментарем знаходяться факти, які вводяться в список фактів за допомогою команди *deffacts,* кількість яких загалом не обмежується.

Наприклад:

```
(deffacts student_list   ;  Список студентів
      (student Ivaniv Ivan)
      (student Petriv Petro) )
```

Факти, задані конструктором deffacts, вводяться в CLIPS за допомогою команди *reset*, яка спочатку видаляє всі факти із списку фактів, а потім вводить факти з існуючого оператора *deffacts.*

В пам'яті системи може бути декілька списків фактів, що автоматично додаються, і всі вони будуть внесені до списку.

Наприклад, якщо ми введемо два таких списки фактів:

```
(deffacts ListNumber1 ; ім'я першого списку фактів
        (code 1)
```

(temp 10)

(state working))

(deffacts listNumber2 ; *ім'я другого списку фактів*

(usemargin TRUE)

(defaultcolor black))

то при виконанні команди *reset* (скидання середовища) в пам'ять системи будуть додані 5 фактів:

1. (code 1)
2. (temp 10)
3. (state working)
4. (usemargin TRUE)
5. (defaultcolor black)

Вигляд результатів проведення вказаних операцій показано на рис. 6.

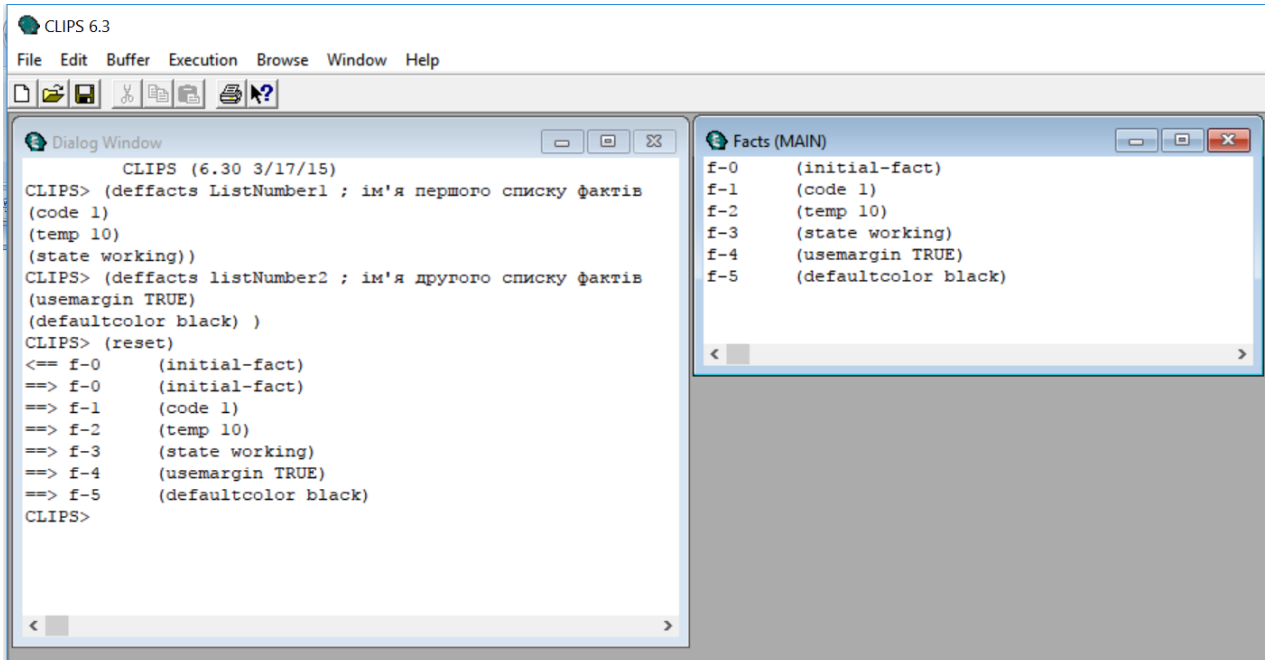

Рис. 6. Вигляд вікна CLIPS після введення списків фактів та виконання команди *reset* .

Аналогічно може бути заданий і масив невпорядкованих фактів. Для цього потрібно спочатку за допомогою конструктора *deftemplate* задати шаблон факту, а потім, використовуючи конструктор *deffacts*, ввести відповідні факти. Розглянемо процес введення на прикладі. Створимо шаблон

```
(deftemplate person
(slot name) (slot sex) (slot birth-year) (slot mother) (slot father)
(multislot illness) (multislot address))
```

та факти:

```
 (deffacts person
(person (name Tamara) (sex female) (birth-year 12-2-1949) (mother Antonina)
(father Petro)(address Lviv))
(person (name Andrij)  (sex male) (birth-year 15-4-1976) (mother Ivanna) (father
Sergij)(address Kyiv))).
```

Після послідовного їх введення в CLIPS та введення команди *reset* в пам’ять системи буде введено два нових факти (рис. 3.7).

Окрім явно заданих конструктором *deffacts* фактів, кожного разу при виконанні команди *reset* CLIPS автоматично додає в список ще й зумовлений факт *initial-fact* як факт з індексом f-0. Факт *initial-fact* може використовуватися для визначення моменту запуску механізму логічного виведення, а також неявно присутній в правилах, для яких не задана передумова (правила, що не мають умовних елементів, у CLIPS автоматично перетворяться в правила з умовою, що перевіряє наявність факту *initial-fact*).

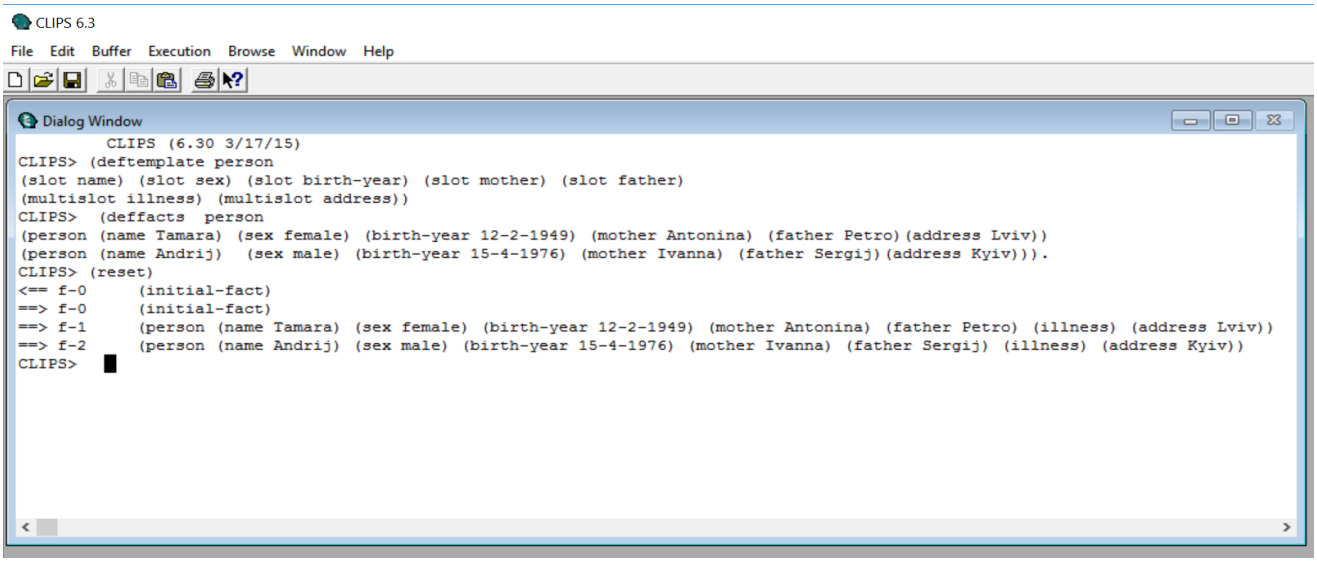

Рис. 3.7. Результати введення в CLIPS шаблона та списку невпорядкованих фактів та виконання команди *reset* .

Факти, додані за допомогою конструктора *deffacts*, можуть використовуватися й видалятися так само, як і будь-які інші факти, додані в базу знань користувачем або програмою за допомогою команди *assert*; відповідно, і переглядати їх та спостерігати за їх перетвореннями можна за допомогою тих же засобів.

Введений конструктором *deffacts* масив фактів можна видалити з бази командою *undeffacts* з синтаксисом:

(undeffacts <deffacts-name>)

3.4. Робота з невпорядкованими (шаблонними) фактами.

Для роботи з шаблонними фактами (з їх вмістом) використовуються функції *modify, duplicate*.

Функція *modify*.

Модифікувати значення слотів невпорядкованих фактів можна за допомогою команди *modify*. Синтаксис команди *modify* є таким:

(modify <fact-index><slot-modifier>+).

У цій команді параметр <slot-modifier>має наступний вигляд:

(<slot-name><slot-value>),

тобто після імені слота повинно бути введено його нове значення.

Команда *modify* діє за принципом витягання первинного факту і подальшого додавання нового факту після модифікації вказаних значень слотів. Інакше, за механізмом робота функції *modify* аналогічна послідовному виконанню двох функцій – видалення (аналогічно до дії, яку виконує команда *retract*) і додавання факту (аналогічно до дії, яку виконує команда *assert*). У зв'язку з цим для модифікованого факту виробляється новий індекс.

У випадку вдалого виконання функція повертає новий індекс модифікованого факту. Якщо ж в процесі виконання сталася помилка, то користувачеві виводиться відповідне повідомлення й функція повертає значення FALSE. Наприклад, якщо заданий факт відсутній у списку фактів (факту зі вказаним індексом нема в списку фактів або ж в шаблоні заданого факту відсутній слот, значення якого потрібно змінити), CLIPS виведе відповідне повідомлення.

За один виклик команда *modify* дозволяє змінювати тільки один факт.

Розглянемо операцію модифікування факту на прикладі. Для цього виконаємо послідовно команду *clear*, створимо шаблон *deftemplate weather*, введемо невпорядкований факт *weather* та команду *modify* модифікування факту f-1:

```
(clear)
(deftemplate weather
        (slot temperature)
        (slot windspeed) )
(assert
(weather (temperature 10) (windspeed 0)) )
```

(modify 1 (temperature 20))

Результатом виконання команди (*modify* 1) буде виведення зі списку фактів факту f-1 та введення в нього модифікованого факту f-2 (див. рис. 8).

Приведені вище команди визначають шаблон *weather*, додають в систему факт про температуру і швидкість вітру, а потім командою *modify* проводяться зміни значення слота *temperature*.

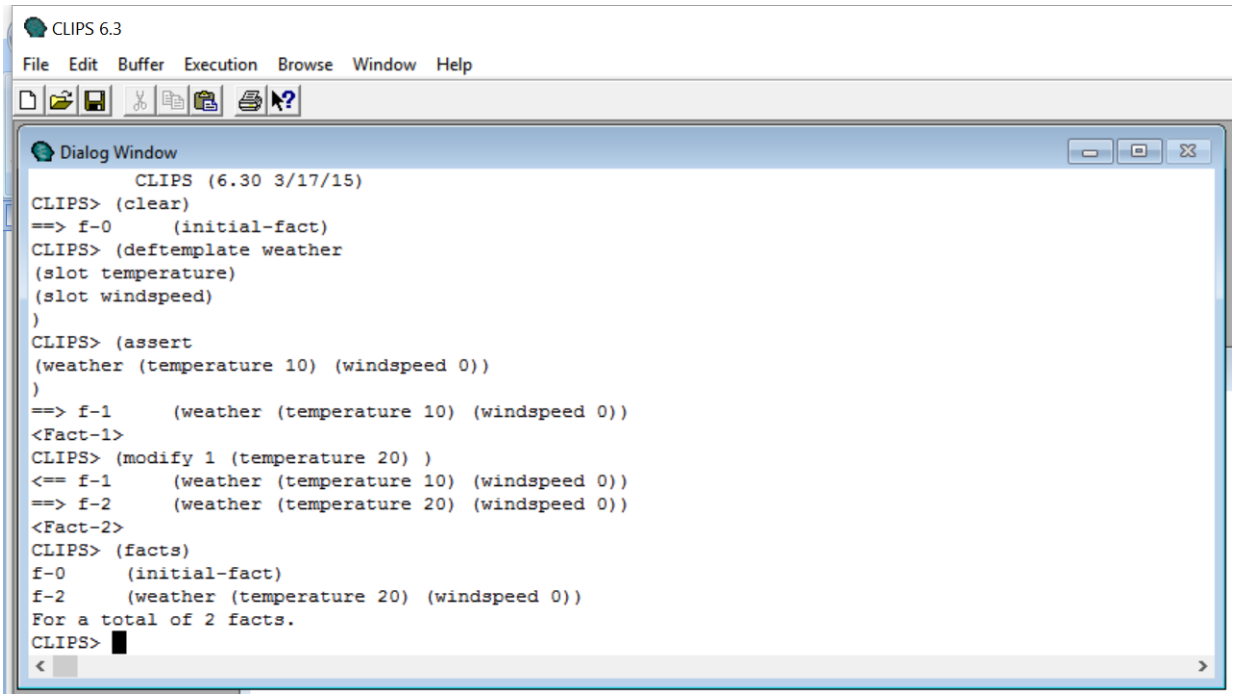

Рис. 8. Результати введення та модифікування факту в CLIPS

Функція *duplicate*.

Команда *duplicate* діє за таким же принципом, що і команда *modify* – за винятком того, що видалення первинного факту тут не відбувається – і дозволяє копіювати існуючі факти за заданим шаблоном, замінюючи вказані користувачем значення слотів.

Синтаксис команди:

(duplicate <fact-index><slot-modifier>+),

де після визначення факту (його індексу) вводиться список з одного або більше нових значень слотів (<slot-modifier>+) зазначеного шаблона.

Фактично, функція *duplicate* створює новий (з новим індексом) невпорядкований факт заданого шаблону шляхом копіювання в нього у незміненому вигляді групи слотів уже існуючого факту та модифікуючи значення вказаного в команді слота (слотів).

У випадку, якщо доданий з допомогою *duplicate* факт уже є у списку фактів, у діалоговому вікні буде повернуте значення FALSE. Факт при цьому доданий не буде. Однак, у CLIPS передбачена також і можливість створення дублікату уже існуючого факту – для цього потрібно дозволити існування однакових фактів у базі знань, встановивши відповідний прапорець у вікні Execution Options.

Результат виконання команди (duplicate 1) у випадку заборони і дозволу існування однакових фактів у базі знань показаний на рис. 9.

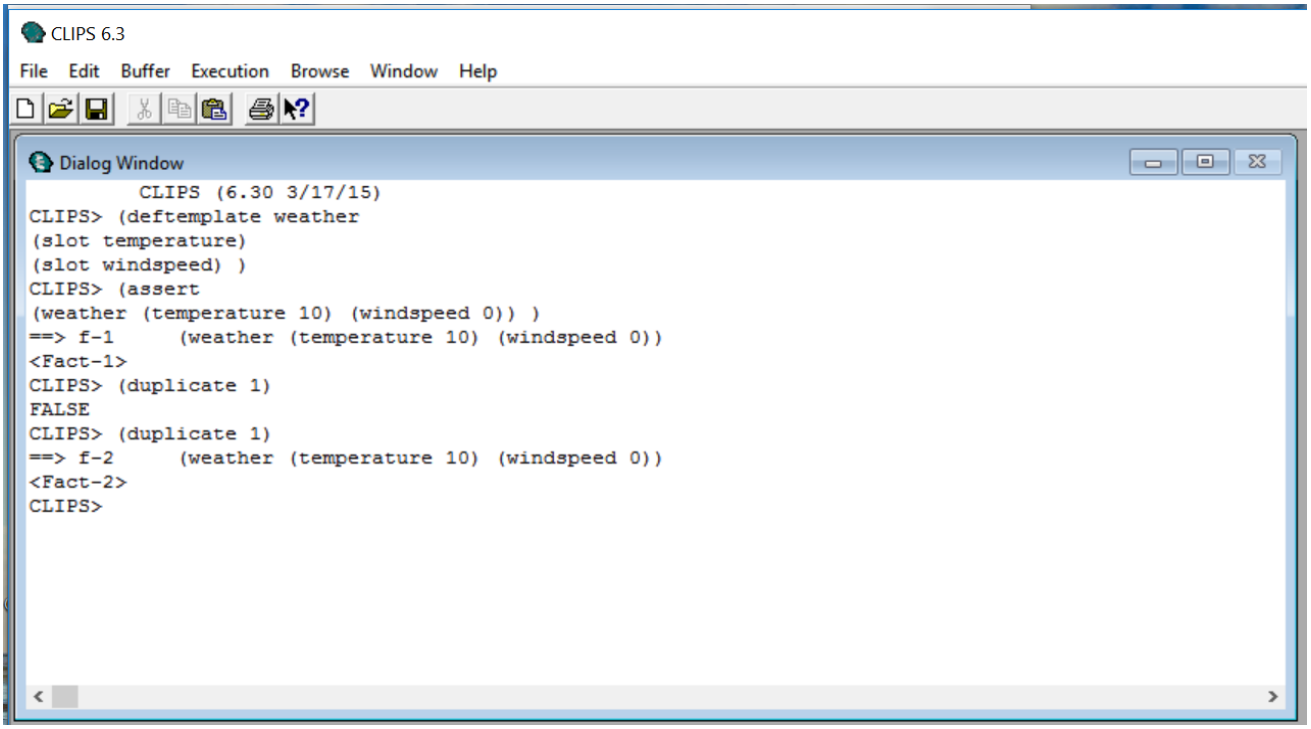

Рис. 9. Результат виконання CLIPS команди (duplicate 1) у випадку заборони і дозволу існування однакових фактів у базі знань.

3.5. Використання невпорядкованих фактів (шаблонів).

Як було відмічено вище, шаблон факту складається з імені факту і визначення полів для зберігання даних, які називаються слотами. Шаблони корисно використовувати для опису будь-якого об'єкту, що має набір атрибутів. Наприклад, якщо перед нами стоїть мета внести в систему інформацію про автомобіль, то його слотами можуть в найпростішому випадку бути колір і марка.

Розглянемо приклад шаблону для фактів, що описують автомобіль. Перейдемо у діалогове вікно CLIPS. Введемо команду *clear* для очищення середовища і введемо відповідний шаблон *deftemplate*:

(deftemplate car "*Шаблон для опису автомобілів*"
(slot color)
(slot model)
(multislot owner))

В даному прикладі ми створили абстрактний шаблон з іменем *car*, в якому є 2 простих слоти для зберігання кольору і моделі автомобіля і один складений слот для зберігання даних про власника.

У разі успішного створення шаблону, CLIPS повернеться в режим очікування введення без яких-небудь повідомлень; в протилежному разі ми отримаємо повідомлення про помилку.

Для перегляду списку шаблонів в системі на поточний момент можна скористатися візуальним інструментом "Deftemplate Manager", який знаходиться в меню "Browse". Відкривши "Deftemlate Manager", можна переконатися, що після введення створеного нами шаблону в списку шаблонів, поряд з автоматично створеним системою шаблоном *initial-fact* є і шаблон *car.* Є також можливість (натиснувши кнопку Pprint у вікні "Deftemlate Manager" (рис. 10)) вивести вигляд шаб-

лону в діалогове вікно.

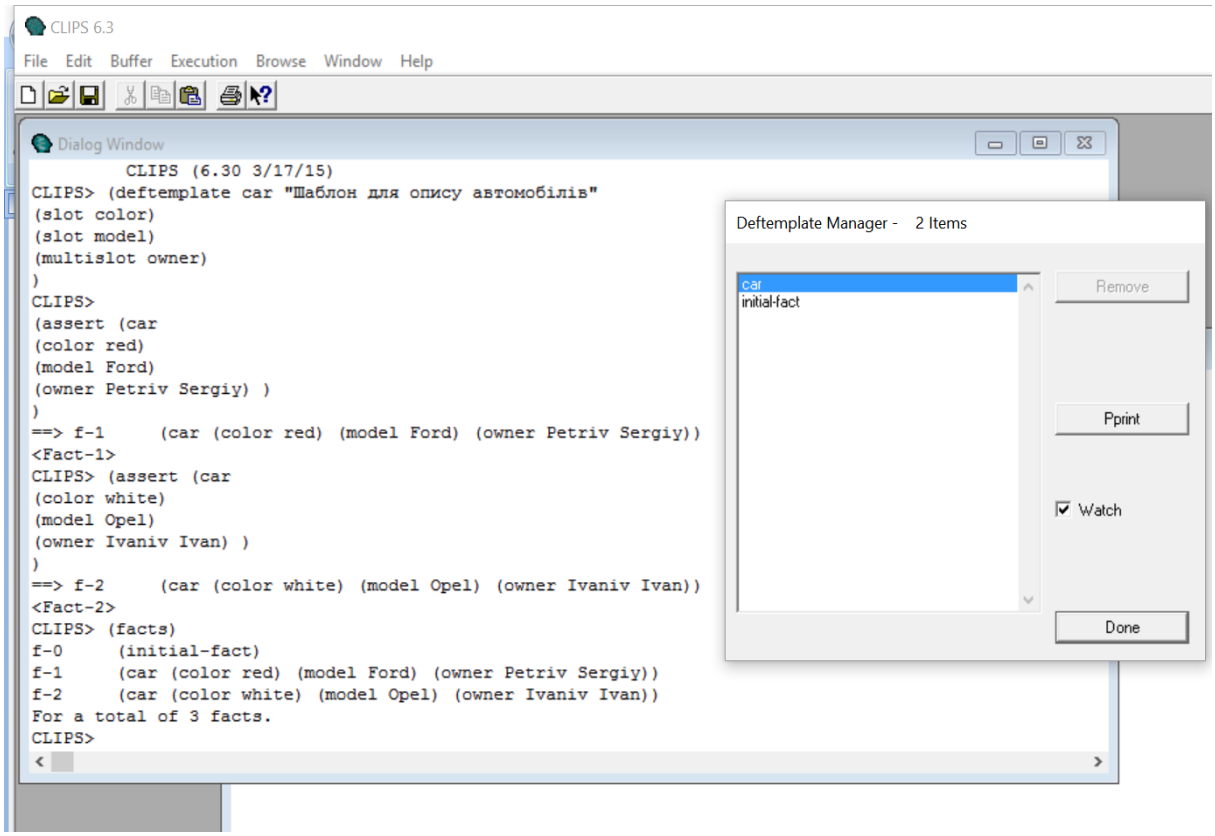

Рис. 10. Результати введення створеного шаблону *car* і двох фактів у діалоговому вікні і вікні менеджера шаблонів.

Після визначення шаблону *car* можна внести інформацію про будь-яку кількість автомобілів у вигляді відповідних неврегульованих фактів. Кожен такий факт визначається шляхом вказівки імені його шаблону (у нашому випадку – *car*) із подальшим заданням імен і значень слотів.

Далі введемо потрібні невпорядковані факти за допомогою функції *assert* (потрібно пам’ятати, що CLIPS – мова, залежна від регістру):

```
(assert (car
        (color red)
        (model Ford)
        (owner Petriv Sergiy) ) )
```

```
(assert (car
            (color white)
            (model Opel)
            (owner Ivaniv Ivan) ) )
```

В результаті отримаємо, що в список фактів CLIPS введені три факти (див. рис. 10):

```
f-0    (initial-fact)
f-1    (car (color red) (model Ford) (owner Petriv Sergiy))
f-2    (car (color white) (model Opel) (owner Ivaniv Ivan))
For a total of 3 facts.
```

3.6. Особливості застосування команд *modify, reset, clear.*

Існує множина додаткових функцій для роботи з фактами (найбільш поширені функції CLIPS можна побачити в Додатку 1). Їх опис і приклади використання можна знайти в on-line документації, а також в деяких з рекомендованих нижче літературних джерел.

Найчастіше під час роботи використовуються команди *modify, reset, clear*.

Для зміни та дублювання шаблонних фактів служать команди *modify, duplicate*. Параметрами цих команд (див. 4) є визначення факту (змінна – адреса або індекс факту), а також нові значення певних, вказаних користувачем, слотів.

Потрібно пам’ятати, що:

- при виконанні команди *modify* індекс модифікованого факту змінюється, а факт, який модифікується, видаляється зі списку фактів;
- при виконанні команди *duplicate* до списку існуючих фактів додається новий факт з новим індексом;
- команда *reset* очищає список фактів, а потім додає в нього факти, оголошені

конструкторами *deffacts*, включаючи факт f-0 (initial-fact); зазвичай використовується у поєднанні з командою *run* для перезапуску написаної програми;

- на відміну від команди *reset*, команда *clear* виконує повне очищення виконуваного середовища і не додає у пам'ять системи ніяких фактів, окрім *initial-fact*; при цьому система очищається не тількифактів, але також і від всіх списків, правил, змінних, шаблонів, які були в неї введені програмно або користувачем до моменту застосування команди *clear*.

Порядок виконання роботи.

1. Ознайомтеся з особливостями опису, створення та введення впорядкованих і невпорядкованих фактів у CLIPS та маніпуляцій з ними.
2. Створіть список впорядкованих фактів у вигляді списку студентів групи у вигляді (student <name><date of birth><kurs>).
3. Використовуючи команду *assert*, введіть створений список впорядкованих фактів у CLIPS, додаючи їх по черзі в діалоговому вікні.
4. Перегляньте список введених фактів, використовуючи команду *facts.*
5. Перегляньте список введених фактів, використовуючи "Deftemplate Manager" з меню "Browse".
6. Перегляньте список введених фактів, використовуючи опцію 1 Facts Windows з меню Windows.
7. Видаліть декілька фактів зі сформованого списку, використовуючи функцію *retract.*
8. Використовуючи функцію *fact-existp*, переконайтеся в існуванні фактів з вказаними індексами.
9. Введіть у пам'ять масив впорядкованих фактів, використовуючи конструктор *deffacts.* Для цього:
   - відкрийте вікно вбудованого редактора в CLIPS;
   - напишіть код введення масиву фактів, використовуючи конструктор

deffacts;

- запишіть файл в пам’ять під іменем lab3-0-facts;
- завантажте файл в CLIPS за допомогою команди *load;*
- впевніться у правильності синтаксису коду введення масиву;
- перегляньте список введених фактів.

10. Пересвідчіться виконання команд, описаних у пп.4-8.
11. Створіть список з 20 невпорядкованих фактів у вигляді списку студентів групи у вигляді:

    (student (name <П_І_Б>) (grupa <група>) (date <дата народження>)),

    використовуючи конструктор *deftemplate.* Для цього:
    - відкрийте вікно вбудованого редактора CLIPS;
    - використовуючи конструктор *deftemplate,* створіть текстовий файл з описом відповідного шаблона та запишіть файл в пам’ять під певним іменем (напр., lab3-1-template);
    - створіть текстовий файл з 5 фактів вказаного у шаблоні формату і запишіть у файл під новим іменем (напр., lab3-1-facts1);
12. Використовуючи написані у вбудованому редакторі коди, введіть з нього в CLIPS шаблон та створені за його допомогою факти.
13. Перегляньте список введених невпорядкованих фактів, використовуючи діалогове вікно, а також пункт 1 Fact Windows підменю Windows графічного інтерфейсу CLIPS.
14. Модифікуйте декілька з введених невпорядкованих файлів; для модифікації фактів використайте команду *modify***.**
15. Продублюйте декілька записаних невпорядкованих фактів, змінивши поля слотів за допомогою команди *duplicate*.
16. Продублюйте декілька записаних невпорядкованих фактів за допомогою команди *duplicate,* не змінюючи значення жодного слоту. Перевірте результат виконання команди з забороною існування у базі знань ідентичних фактів і з дозволом існування таких фактів.

17. Видаліть декілька невпорядкованих фактів зі сформованого списку, використовуючи функцію *retract*.
18. Перегляньте список невпорядкованих фактів.
19. Запишіть внесені в CLIPS факти в файл під іменем lab3-facts2.
20. Повністю очистіть пам’ять системи.
21. Використовуючи записані раніше в файли конструкцію шаблону та створені відповідні невпорядковані факти, введіть їх в систему за допомогою команди load.
22. Пересвідчіться у отриманні результату, аналогічного з отриманим при введенні з клавіатури або з використанням вбудованого редактора.
23. Написати звіт про виконану роботу; при оформленні звіту використо-вувати скріншоти окремих етапів роботи. В додатку представити роздрук вмісту діалогового вікна CLIPS зі всіма результатами проведеної роботи.

Контрольні запитання:

1. Що таке продукційні системи представлення знань?
2. Яку роль відіграють факти в продукційних експертних?
3. Які види фактів використовуються в CLIPS?
4. Що таке впорядкований факт і чим він відрізняється від невпорядкованого?
5. У чому полягає різниця між впорядкованими і невпорядкованими фактами CLIPS?
6. Як можна ввести впорядковані факти в робочу пам’ять CLIPS?
7. Які маніпуляції можна здійснювати з фактами в CLIPS і які команди для цього використовуються?
8. Які команди використовуються для маніпуляцій з впорядкованими фактами CLIPS?
9. Які команди не можна використовувати для маніпуляцій з впорядкованими фактами CLIPS і чому?
10. Який порядок створення невпорядкованих фактів CLIPS?

11. Що таке *initial-fact*, як і яким чином він утворюється?
12. Що таке конструктор *deffacts* і для чого він призначений?
13. Якою командою можна знову ввести в список факти, які вводяться конструктором *deffacts*?
14. З допомогою якої команди можна ввести в CLIPS масив фактів?
15. В чому полягають особливості використання команд *modify* і *duplicate*?
16. Які особливості використання команд *clear* і *reset* і в яких випадках вони застосовуються?
17. Яка команда дозволяє переглянути список введених фактів?
18. Як можна переглянути список фактів, використовуючи графічний інтерфейс CLIPS?
19. Як можна переглянути список шаблонів невпорядкованих фактів у CLIPS?
20. Чи можна ввести в список фактів CLIPS аналог вже існуючого у ньому?
21. Які можливості дає використання інструментів Watch, Options вікна Execution для роботи з фактами?
22. Які можливості в роботі з фактами надають інструменти вікна Windows?
23. Як і в якому форматі можна записати набраний у вбудованому редакторі код на носій?
24. Як зчитати набраний код з носія в робочу пам’ять CLIPS?

Література:

1. CLIPS Reference Manual. Volume I. Basic Programming Guide. Version 6.30 March 17th 2015 – 416 p. [Електронний ресурс] – Режим доступу: http://clipsrules.sourceforge.net/documentation/v630/bpg.pdf
2. Джарратано Джозеф. Экспертные системы. Принципы разработки и программирование, 4е изд: Пер. с англ. / Джозеф Джарратано, Гари Райли. – М., Изд. дом “Вильямс”, 2007. – 1152 с.
3. Частиков А. П. Разработка экспертных систем. Среда CLIPS. / А. П. Частиков., Т. А. Гаврилова, Д. Л. Белов. – СПб., «БХВ-Петербург», 2003. – 608 с.

Лабораторна робота 4.

Тема: СТВОРЕННЯ ПРАВИЛ В СЕРЕДОВИЩІ CLIPS.

Мета роботи: Отримати навички створення і введення в CLIPS правил як складової частини бази знань і одного з основних методів їх представлення. Створити програму, яка включає низку правил, виконання попереднього правила в якій запускає виконання наступного.

Завдання роботи:

- Засвоїти особливості створення правил у середовищі CLIPS.
- Набути навички в створенні правил, використовуючи конструктор *defrule*, та їх введення в середовищі CLIPS.
- Написати код програми, яка передбачає послідовне виконання низки з чотирьох правил, кожне з яких виконується лише тоді, коли запущене на виконання попереднє.
- Створити програми з умовою виконання першого правила як за допомогою відповідного факту, так і без його введення. Запустити програми та роздрукувати результати їх роботи.
- Запустити виконання програми і пересвідчитися, що послідовність введення правил у CLIPS не впливає на роботу програми, а порядок її виконання визначають наявні в пам’яті системи факти.

Вихідні матеріали:

4. Правила у CLIPS. Деякі особливості застосування правил у CLIPS.

Експертна система (зокрема, ЕС продукційного типу, створена за допомогою CLIPS) зможе виконувати передбачену її призначенням змістовну роботу

тільки в тому випадку, якщо її база знань є достатньою для вирішення поставлених задач, тобто в ній присутні не тільки достатня кількість фактів, але й присутня друга необхідна складова знань – правила.

Правила в CLIPS служать для подання евристик (або ж так званих «емпіричних правил»), які визначають набір дій, які виконуються при виникненні деякої ситуації. При розробці експертної системи її розроблювач визначає і створює набір правил, які можуть бути корисні при вирішенні задач, для розв'язання яких вона повинна застосовуватися.

Як і факти, правила в CLIPS можуть вводитися або безпосередньо, в діалоговому режимі, або ж завантажуватися з файлу з правилами, спеціально створеного за допомогою текстового редактора – або вбудованого, або ж зовнішнього. Мабуть, варто відмітити, що в діалоговому режимі вводити варто лише поодиночні правила або ж невеликі їх масиви; для введення в систему великих масивів правил доцільно використовувати або вбудований у CLIPS редактор (нагадаємо, що він викликається комбінацією клавіш «CTRL+N» або з використанням опції New меню File; при цьому в CLIPS відкривається вікно для введення коду Untitled1 (див. рис.5.2)), або ж будь-який доступний користувачеві текстовий редактор (у т. ч. WordPad чи Microsoft Word). Використання редактора, зокрема, дає можливість оперативно перевизначати потрібні конструкції під час розробки програми; ця можливість є надзвичайно корисною, оскільки дозволяє виправити можливу помилку набору і перевизначити конструкцію за допомогою декількох натиснень клавіш безпосередньо у редакторі. На відміну від введення з командного рядка, де у випадку, коли під час створення якоїсь конструкції буде допущена друкарська помилка, то потрібно буде наново набрати увесь її текст. Написаний код зі створеним масивом правил зазвичай записується в файл, який потім може бути зчитаний в CLIPS.

Після введення коду у вікні вбудованого редактора, його (так же як і при роботі з фактами) можна або записати в файл з розширенням *.clp (або *.txt) для наступного введення в систему зчитуванням (напр., за допомогою команди Load ме-

ню File), або ж безпосередньо ввести в систему, використавши команди меню Edit та Buffer, або ж замаркувавши потрібний код (або його частину) і натиснувши комбінацію клавіш «CTRL+M». Підтвердженням правильності синтаксису введення (тобто відсутності синтаксичних помилок написаного коду) буде поява в діалоговому вікні запрошення CLIPS>.

4.1. Структура правил і особливості їх застосування в CLIPS.

Розробник, ґрунтуючись на особливостях предметної області та поставленої задачі, визначає сукупність правил, які можуть бути використані експертною системою при вирішенні конкретної проблеми, формує їх та вводить у базу правил.

Правила в CLIPS служать для представлення так званих “емпіричних знань”, які визначають набір дій, що виконуються системою при виникненні деякої ситуації, і складаються з передумов і наслідку (дії).

Передумови називаються ще “ЯКЩО-частиною”, лівою частиною правила (або ж LHS – left-hand side) і загалом є набором (з однієї або ж з низки) умов (або умовних елементів), які повинні бути обов’язково задоволені, щоб правило було активоване. Передумови правил задовольняються або ні – залежно від наявності в пам’яті системи певних введених в неї фактів (або деяких спеціально створених екземплярів об’єктів, класів).

Один з найбільш поширених типів умовних виразів в CLIPS – зразки (patterns). Вони складаються з набору обмежень, кожне з яких використовуються для визначення того, чи задовольняє деякий факт умовному елементу правила. Іншими словами, зразок задає деяку “маску” для фактів.

Процес порівняння зразків з фактами називається *процесом зіставлення зразків* (pattern-matching) і здійснюється в CLIPS через вбудований механізм (*механізм логічного виведення* (inference engine)), який автоматично зіставляє зразки з наявним в оперативній пам’яті системи поточним списком фактів з метою пошуку правил, які застосовні на даний момент. Якщо всі передумови правила задоволені, правило активується і стає застосовним до поточної ситуації. Наслідком його за-

стосування є виконання певних дій, набір яких прописаний у т. з. “дійовій” (її ще називають консеквентною) правій (RHS – right-hand side) “ТО-частині” правила.

Таким чином, дії, задані “дійовою” частиною правила, виконуються за командою механізму логічного виведення. У випадку, якщо в якийсь робочий момент застосовними виявиться більше від одного правила (тобто наявними в робочій пам’яті системи фактами задовольняються умовні частини декількох правил, наявних у базі знань, одночасно), виникає конфлікт у черговості їх виконання; для його розв’язання механізм логічного виведення застосовує спеціальну процедуру – *стратегію вирішення конфліктів* (conflict resolution strategy), яка визначає, яке саме правило з присутніх у робочій пам’яті системи буде виконано першим. Після цього CLIPS виконує дії, описані в правій частині обраного правила; внаслідок їх виконання в список фактів додаються нові факти, які, знову ж таки, можуть вплинути на список активованих правил, модифікувати його, – і приступає до вибору наступного правила. Цей процес триває доти, поки список застосовних правил не спорожніє або ж не буде досягнута мета виведення.

В CLIPS виконання програми не вимагає явного визначення її потоку. Знання (правила) і дані (факти) тут розділені, а механізм логічного виведення застосовує наявні в робочій пам’яті факти до знань, активує відповідні їм правилаз бази знань, формує список застосовних в кожний момент роботи правил (т. з. “*агенду*”), визначає за певною процедурою пріоритет кожного правила і потім послідовно їх виконує. Цей процес називається *основним циклом виконання правил* (basic cycle of rule execution).

Термін “виконання правила” означає, що система CLIPS виконує дії одного з правил,обраного з наявних у робочому списку правил. У випадку, коли робочий список правил пустий (правил у ньому нема), програма зазвичай припиняє свою роботу. Якщо правило в списку одне – воно виконується. Якщо ж в робочому списку правил є декілька правил, то система CLIPS автоматично визначає, яке з правил є відповідним для запуску, використовуючи при цьому одну з передбачених в ній спеціальних процедур – стратегій вирішення конфлікту, який у цьому

випадку має місце – зокрема, і з врахуванням пріоритету правила. (Під пріоритетом правила розуміється його властивість, яка задана розробником безпосередньо або вираховується самою системою при його активуванні.) При надходженні у список нового правила сама CLIPS щоразу по-новому, з урахуванням їх пріоритету, упорядковує правила, що знаходяться в робочому списку, і першим запускає виконання правила з найвищим пріоритетом.

4.2. Конструктор *defrule*.

Оскільки без правил не обходиться жодна експертна система, особливості їхнього подання в ній є дуже важливою частиною будь-якої експертної оболонки. Відповідно, в CLIPS також використовується певна система представлення правил, маніпуляцій з ними, а також порядок їх застосування.

Для оголошення і додавання нових правил у базу правил в CLIPS використовується конструктор *defrule* з таким синтаксисом:

(defrule *<rule name>* *[<comment>]*
[<необов'язкове_визначення_властивості_правила>]
<patterns>* *; Ліва частина (Left-hand Side – LHS) правила*
=>
<actions>*); *Права частина (Right-hand Side – RHS) правила*

Розглянемо докладніше особливості написання коду правила.

Все правило повинне бути поміщене в круглі дужки; крім того, в круглі дужки необхідно помістити й кожну з його компонент – зокрема, <patterns> (шаблони) і <actions> (дії). Правило може мати й декілька шаблонів і дій (або ж не мати їх зовсім). Умовна (<patterns>) і дійова (<actions>) частини правила обов'язково розділяються знаком =>.

Оскільки зазвичай круглі дужки, в які поміщають шаблони і дії, є вкладе-

ними, то кількості відкриваючих і закриваючих круглих дужок кожного правила повинні бути правильно збалансовані.

Заголовок правила складається з трьох частин (полів).

Правило завжди починається з ключового слова *defrule*,за яким слідує поле імені правила. *Ім'я правила* повинне бути значенням типу *symbol*; воно не може бути зарезервованим словом мови CLIPS і повинне бути унікальним – потрібно пам'ятати, що у разі повторного визначення правила з іменем, яке уже існує в базі правил, старе правило буде видалене з неї, навіть якщо нове неможливо буде додати внаслідок синтаксичної помилки або через будь-яку іншу причину.

За іменем правила слідує взятий у лапки рядок *коментаря*, який є необов'язковим – зазвичай в ньому описують призначення правила або надають будь-яку іншу інформацію, яку розробник вважає за потрібне ввести для кращого розуміння коду. Коментар також повинен бути значенням типу *string*; значення коментаря зберігається разом з правилом. Нагадаємо, що, на відміну від звичайних коментарів, що починаються з крапки з комою, коментар, який слідує за іменем правила, не ігнорується і може бути виведений на зовнішній пристрій разом з рештою правила за допомогою команди *ppdefrule.*

За заголовком правила слідує визначення його властивості, яке також є необов'язковим; особливості написання і застосування цього елементу правила будуть розглянуті нижче.

Далі слідує умовна (ліва, або LHS, чи антецедентна) частина правила, яка складається з окремих умовних елементів CE (CE – Conditional Element), які повинні задовольнятися для активування правила. Загалом, обмежень на кількість таких елементів у правилі та їх тип нема – їх може взагалі не бути (тобто умовний елемент у деяких правилах може бути відсутнім), а може бути й будь-яка потрібна їх кількість та потрібна їх комбінація.

Якщо правило не має передумов (тобто умовна частина правила пуста), то в якості умови для даного правила в CLIPS розуміється спеціальний шаблон *initial-fact*. Оскільки факт з ім'ям *initial-fact* поміщається в пам'ять системи після входу в

неї або після очищення пам'яті (командами *clear* і *reset*) автоматично як факт з іменем f-0, то і всі правила, що не мають шаблонів в своїх лівих частинах, знову ж таки автоматично активуються після виконання CLIPS цих команд. Таким чином, після включення системи або виконання команд *clear* чи *reset* в робочий список активованих правил автоматично поміщаються усі правила, що не мають умовних елементів (або шаблонів) у своїй лівій (умовній, або антецедентній) частині.

Останньою частиною правила є *дія* (або *права частина правила*), яка містить список дій, що виконуються при виконанні правила механізмом логічного виведення; кількість дій у правилі загалом не обмежена. Дії правила виконуються послідовно, але тоді і тільки тоді, коли всі умовні елементи в лівій частині цього правила задоволені.

Правило може і не мати дій (такий спосіб використання правил також іноді застосовується) – якщо в правій частині правила не визначене жодне дія, правило може бути активоване й виконане, але при цьому нічого не відбудеться.

Отже:

- ліва частина правила є низкою умов (умовних елементів, фактів), які повинні виконуватися, щоб правило було застосовне;
- у CLIPS прийнято вважати, що умова виконується, якщо відповідний зразку у LHS факт (факти) присутній (присутні) у списку фактів;
- для розділення лівої і правої частин правил використовується символ =>;
- права частина правила є сукупністю дій, які повинні бути виконані, якщо правило запущене на виконання.

4.3. Пріоритет правила

Під пріоритетом правила розуміється певний цілочисельний атрибут, який ще прийнято називати значущістю(*salience*) правила. Значущість правила входить у конструктор *defrule* як необов'язковий елемент <*визначення властивості правила*>, який поміщається між його іменем і шаблоном умов.

Елемент правила, який визначає його властивості,складається з ключового слова *declare* і наступної за ним вказівки властивості.

Синтаксис визначення властивості правила в CLIPS:

<*визначення-властивості-правила*> = (declare<*властивість-правила*>)

<*властивість-правила*> = (salience <*цілочисельний вираз*>)

або (auto-focus <*логічний вираз*>)

У правила може бути дві властивості – *salience* і *auto-focus*.

Властивість *salience* дозволяє користувачеві призначати потрібний пріоритет для своїх правил. Оголошуваний пріоритет може набувати цілочисельні значення в діапазоні від -10 000 до +10 000.

Для визначення цілочисельного значення пріоритету також може бути використаний будь-який математичний вираз, що представляє пріоритет правила; для його визначення також можна використовувати глобальні змінні і функції, за допомогою яких вираховується його значення. Проте, краще не використовувати в цьому виразі функцій, що мають побічну дію.

Властивість *auto-focus* використовується при багатомодульній архітектурі побудови програм і може приймати два логічних значення – TRUE або FALSE. Якщо значенням властивості *auto-focus* є TRUE, то команда *focus* автоматично виконується щоразу при активуванні правила у модулі, у якому визначене дане правило. Якщо ж це значення – FALSE, то при активуванні правила не відбувається ніяких дій. За замовчуванням, у CLIPS значення властивості *auto-focus* встановлене як FALSE.

Значення пріоритету може обчислюватися:

- при додаванні нового правила;
- при активуванні правила;
- на кожному кроці основного циклу виконання правил.

Два останні зазначені варіанти задавання значення пріоритету називаються *динамічним пріоритетом* (dynamic salience). За замовчуванням, значення пріоритету в CLIPS обчислюється тільки під час додавання правила. Для зміни цієї уста-

новки можна використовувати вікно Options з меню Execution; у вікні Execution Options у вказують вибраний режим обчислення пріоритету, вибираючи одне з трьох значень випадного списку Salience Evaluation (рис. 4.1):

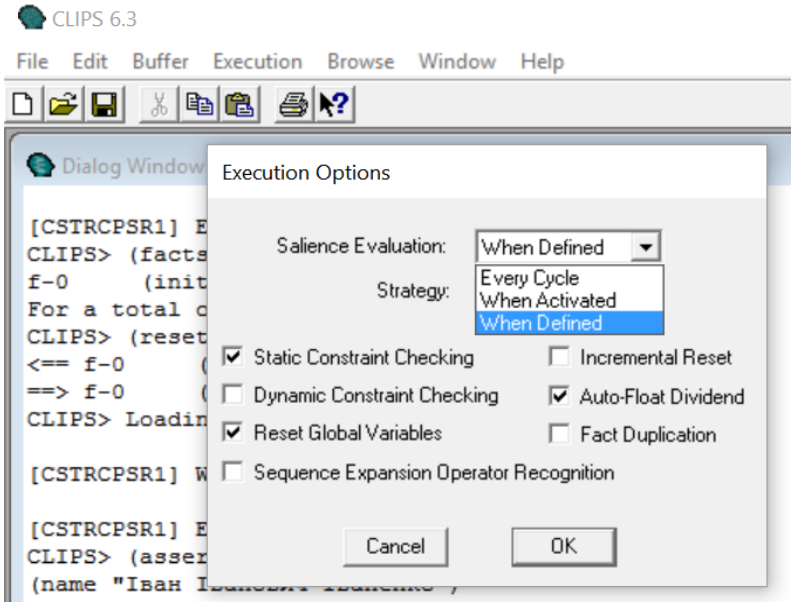

Рис. 4.1. Вікно налаштування параметрів механізму логічного виведення з відкритим списком режимів обчислення пріоритету правил в CLIPS

Every Cycle;

When Activated;

When Defined.

У випадку, якщо пріоритет правила явно не заданий, йому привласнюється значення за замовчуванням – 0. Чим більше число, що визначає пріоритет правила, тим вищий його пріоритет.

4.4. План рішення задачі.

Процес зіставлення фактів і зразків виконується блоком виведення CLIPS, який автоматично зіставляє зразки, виходячи з поточного стану списку фактів, і

визначає, які з правил зі списку є застосовними. Якщо всі умови правила виконуються, то воно активується і поміщається в список активованих правил, який є *планом рішення задачі* і називається *агендою*.

Загалом, коли активується нове правило, воно розміщується в плані рішення задачі (списку активованих правил), керуючись наступним:

1. Тільки що активоване правило поміщається вище від всіх наявних у списку правил з меншим пріоритетом і нижче від всіх правил з більшим пріоритетом.
2. Для визначення розміщення нового правила серед інших правил з однаковим пріоритетом використовується певна стратегія вирішення конфліктів.
3. Якщо для активованого правила за допомогою кроків 1 і 2 не можна визначити його порядок в списку активованих правил, то це правило упорядковуються разом з іншими правилами довільним чином.

Список правил, що знаходяться в робочому списку правил, можна вивести на зовнішній пристрій за допомогою команди *agenda* з синтаксисом:

(agenda [<module-name>])

Якщо <module-name> не вказано, то відображаються всі активовані правила поточного модуля. Якщо <module-name> вказане, то відображаються всі активації зазначеного модуля. Якщо <module-name> – символ «*», то відображаються активовані правила у всіх програмах у всіх модулях.

Якщо в робочому списку правил активовані правила відсутні, то після введення команди *agenda* знову з’являється запрошення CLIPS.

Окрім виведення списку активованих правил в робоче вікно шляхом виконання команди *agenda*, у Windows-версії CLIPS передбачені ще два варіанти його перегляду – в окремому вікні Agenda (MAIN) (для використання цього інструменту потрібно скористатися пунктом Agenda Window з меню Window), та у вікні Agenda Manager (Менеджер плану рішення задачі; для цього потрібно вибрати пункт Agenda Manager з меню Browse).

Вигляд діалогового вікна CLIPS з результатами виконання команди *agenda,*

введеної з командного рядка, та випадних вікон Agenda (MAIN) та Agenda Manager показаний на рис. 2.

Потрібно відмітити, що інструмент Agenda Manager, на відміну від перегляду за допомогою Agenda (MAIN), надає можливість в разі потреби здійснювати корекцію плану рішення задачі шляхом видалення із нього окремих активованих правил або ж запускати правила в будь-якому довільному порядку, у т. ч. і здійснити покрокове її виконання. Для цього потрібно вибрати відповідне правило зі списку активувань та натиснути кнопку Remote або Fire, відповідно (рис. 4.2).

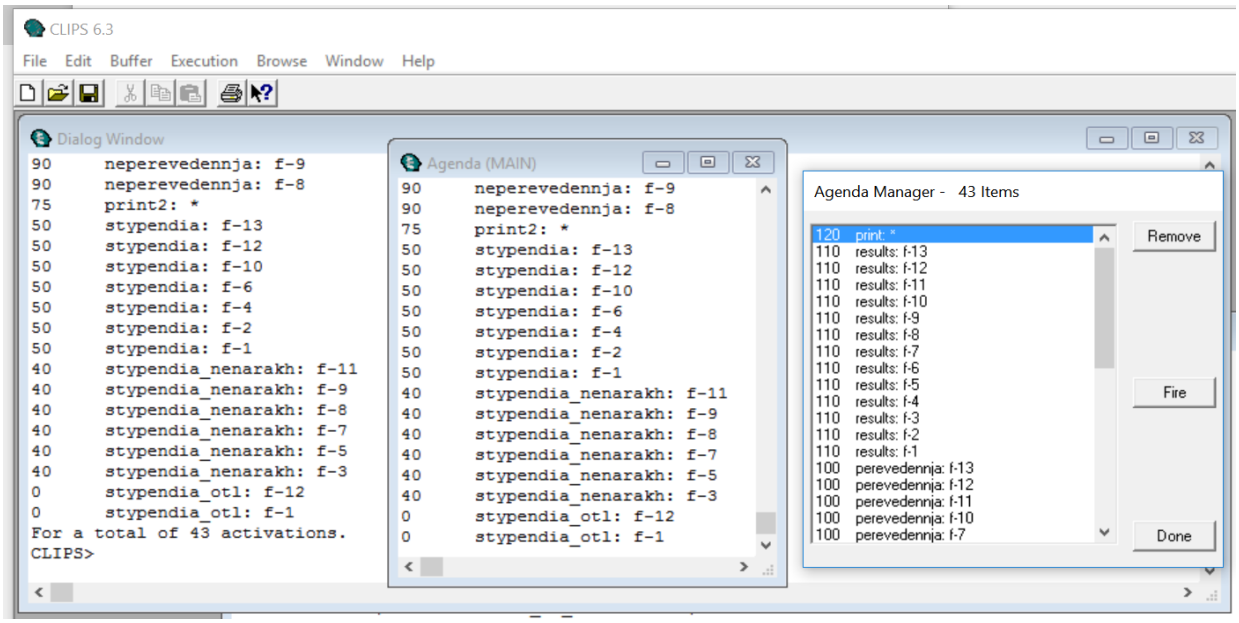

Рис. 2. Вигляд діалогового вікна CLIPS з результатами виконання команди *agenda*, введеної з командного рядка, та випадних вікон Agenda (MAIN) та Agenda Manager.

4.5. Стратегії вирішення конфліктів у CLIPS.

Дії, описані в застосовних правилах, виконуються тоді, коли блок виведення CLIPS отримує команду почати виконання застосовних правил. Якщо в поточному плані рішення задачі є багато застосовних правил, то для того, щоб вибрати одне правило, дії якого повинні бути виконані, або для визначення черговості їх

виконання механізм виведення використовує стратегію вирішення конфліктів. Для цього в CLIPS передбачене використання семи різних стратегій вирішення конфліктів.

Стратегія глибини (depth strategy) – тільки що активоване правило розміститься вище від всіх правил з таким же пріоритетом.

Стратегія ширини (breadth strategy) – тільки що активоване правило розміститься нижче від всіх правил з таким же пріоритетом.

Стратегія спрощення (simplicity strategy) – між всіма правилами з однаковим пріоритетом тільки що активовані правила розміщаються вище від усіх активованих правил з рівною або більшою *визначеністю* (specificity), під якою розуміють кількістю зіставлень, які потрібно зробити в лівій частині правила для його активування. Кожне зіставлення з константою або заздалегідь пов'язаною з фактом змінною додає до визначеності одиницю.

Стратегія ускладнення (complexity strategy) – серед правил з однаковим пріоритетом тільки що активовані правила розміщаються вище від усіх раніше активованих правил, визначеність яких є рівною або меншою від визначеності щойно активованого.

LEX (LEX strategy) – для визначення місця активованого правила в плані рішення задачі використовується «новизна» факту, що активував правило. CLIPS маркує кожен факт тимчасовим тегом для відображення відносної новизни появи кожного факту в системі. Правило, активоване новішими фактами, розташовується перед правилом, активованим більш пізніми зразками. Вище від усіх інших у плані рішення задачі поміщується правило з відповідним найбільшим тимчасовим тегом.

MEA (MEA strategy) – на відміну віл стратегії LEX, в стратегії MEA порівнюються тільки тимчасові теги перших зразків двох активувань; правило з більшим тегом поміщається в план рішення задачі перед правилом з меншим. Якщо ж обидва активовані правила мають однакові тимчасові теги, асоційовані з першим зразком, то для визначення розміщення правила в плані рішення задачі викорис-

товується стратегія LEX.

Випадкова стратегія (random strategy) – кожному активованому правилу призначається випадкове число, що використовується для визначення місця розташування серед активувань з однаковим пріоритетом. Це випадкове число зберігається при зміні стратегій – таким чином, той же порядок відтворюється при наступній установці випадкової стратегії.

Необхідна стратегія може бути вказана вибором відповідної опції в вікні Execution Options (рис. 4.3). За замовчуванням, в CLIPS встановлена стратегія глибини *Depts*.

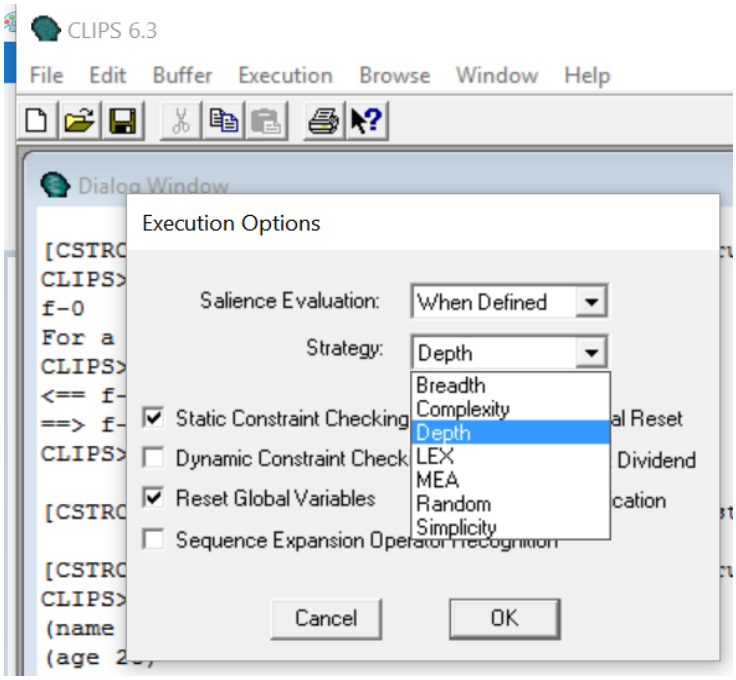

Рис. 4.3. Вікно з розкритим списком стратегій вирішення конфліктів.

4.6. Алгоритм виконання правил (логічне виведення) в CLIPS

Як уже згадувалося вище, в якості механізму логічного виведення CLIPS використовує прямий ланцюжок міркувань. Після того, як у пам’ять системи введені всі необхідні правила і приготовлені початкові списки фактів, CLIPS готовий виконувати правила. Як і в усіх продукційних системах, знаходження вирішення

поставленої задачі в CLIPS полягає у циклічному виконанні певних дій, які мають назву основного циклу виконання правил.

Основний цикл виконання правил у CLIPS полягає у наступному.

На відміну від традиційних мов програмування, в яких точка входу, точка зупинки і послідовність обчислень явно визначаються програмістом, у CLIPS потік виконання програми абсолютно не вимагає такого визначення. Знання (правила) і дані (факти) є розділені (в базу правил БП і базу фактів БФ, відповідно), і механізм логічного виведення, що вбудований в CLIPS, застосовує наявні у пам'яті дані до знань, формуючи за певною прийнятою стратегією вирішення конфліктів список застосовних правил; після формування такого списку система послідовно виконує їх.

Розглянемо послідовність дій (кроків), що виконуються системою CLIPS в циклі під час виконання програми (див. рис. 4.4.):

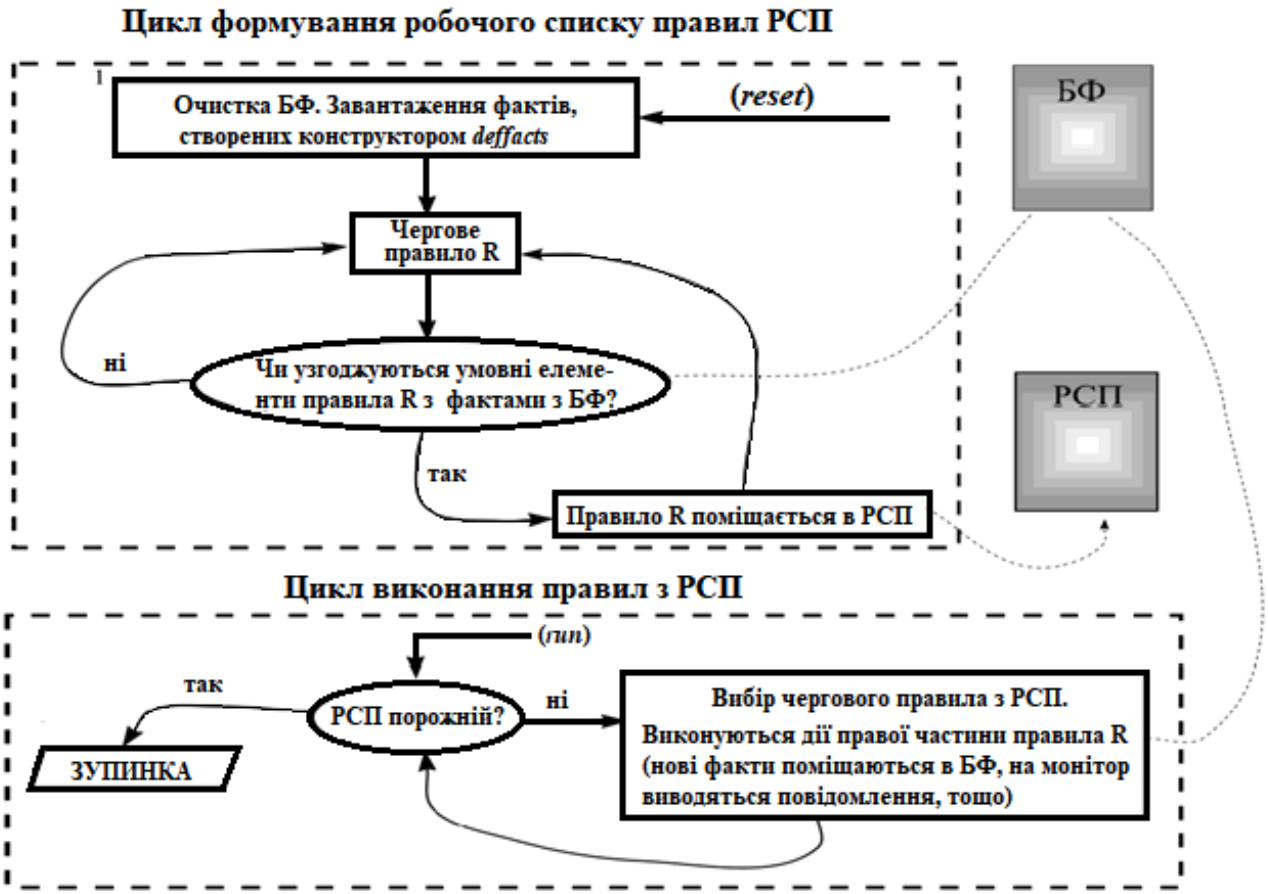

Рис. 4.4. Цикл обробки правил у CLIPS

1. Відразу ж після введення правил і фактів в пам'ять, CLIPS робить спроби зіставити умовні елементи в лівій частині правила з наявними в робочій пам'яті (РП) фактами. Якщо все умовні елементи правила узгоджуються з фактами, правило активується і поміщається в т. з. робочий список правил (РСП), який іноді називають планом рішення задачі. В РСП може перебувати будь-яка кількість правил – від нуля і більше.

2. Активування правил ще не означає їх виконання. Логічне виведення в CLIPS починається по команді *run*. В його ході відбувається запуск першого правила з наявного в РП на поточний момент робочого списку правил. Запуск правила полягає у виконанні всіх дій, описаних в правій частині правила.

В результаті цих дій отримуються нові знання (факти), які, якщо вони не є цільовими, поміщаються в РП системи. В результаті чого можуть бути активовані нові правила, які знову поміщаються в РСП. Розміщення правила в РСП визначається його пріоритетом (salience) і поточною стратегією вирішення конфліктів (ці поняття докладніше будуть описані нижче). Деякі правила в результаті виконання дій можуть бути деактивовані. У цьому випадку вони видаляються з РСП.

Таким чином, після запуску команди *run* в модулі логічного виведення CLIPS фактично «прокручуються» два цикли (див. рис. 5.3). У першому циклі відбувається аналіз завантажених правил з метою визначення тих з них, ліві частини яких узгоджуються з фактами з БФ; такі правила активуються і поміщаються в РСП. У другому циклі відбувається послідовне виконання правил з РСП.

4.7. Виконання програм в CLIPS.

Програма у CLIPS може бути викликана на виконання за допомогою команди *run*, яка має синтаксис:

(run [<limit>])

У цьому визначенні необов'язковий параметр *limit* указує кількість правил,

що підлягають запуску; після запуску останнього з вказаної цим параметром кількості правил їх виконання системою припиняється. Якщо параметр *limit* не заданий або його значення рівне -1, то запуск правил відбувається до тих пір, допоки в робочому списку правил не залишиться жодного правила.

Правило активується після того, як всі шаблони правила узгоджуються з наявними в списку системи фактами. Процес зіставлення з шаблонами під час роботи CLIPS завжди підтримується в актуальному стані і відбувається без урахування того, коли відбулося введення фактів в список фактів – до або після визначення того або іншого правила.

Оскільки для виклику правил на виконання потрібні факти, необхідні механізми їх надходження в оперативну пам'ять системи, у т. ч. і при її очищенні. Одним з методів забезпечення запуску або перезапуску експертної системи, передбачених у CLIPS, є виконання команди *reset*, дією якої спочатку зі списку фактів видаляються усі факти, а потім заново вводяться в нього факти, що були введені у систему всіма конструкторами *deffacts,* актуальними на момент виконання цієї команди. Якщо ж факти були введені в список іншим чином (напр., за допомогою команди *assert*), то їх потрібно ввести заново.

Якщо факти, введені в список фактів (за допомогою команди *reset* чи повторним їх введенням), задовольняють шаблонам одного або декількох правил, то це приводить до автоматичної передачі цих правил в робочий список правил. При введенні активованого правила в робочий список CLIPS автоматично запускає механізм визначення його пріоритету та визначення місця в плані рішення задачі. Потім, після введення команди *run*, починається виконання програми. Результати виконання цих команд висвітлюються в діалоговому вікні. Вигляд виведених в діалоговому вікні результатів виконання команд (насамперед – наявність у ньому даних трасування і їх повнота) залежить від параметрів, які визначаються у вікні Watch Options (рис. 4.5).

Спроба повторно виконати команду *run* не приведе до запуску програми та, відповідно, й до отримання результатів її роботи. Перевірка робочого списку пра-

вил у цьому випадку покаже, що запуск правил не відбувся не тому, що в робочому списку правила та факти відсутні. Для повторного запуску програми потрібно знову ввести факти і відновити правило.

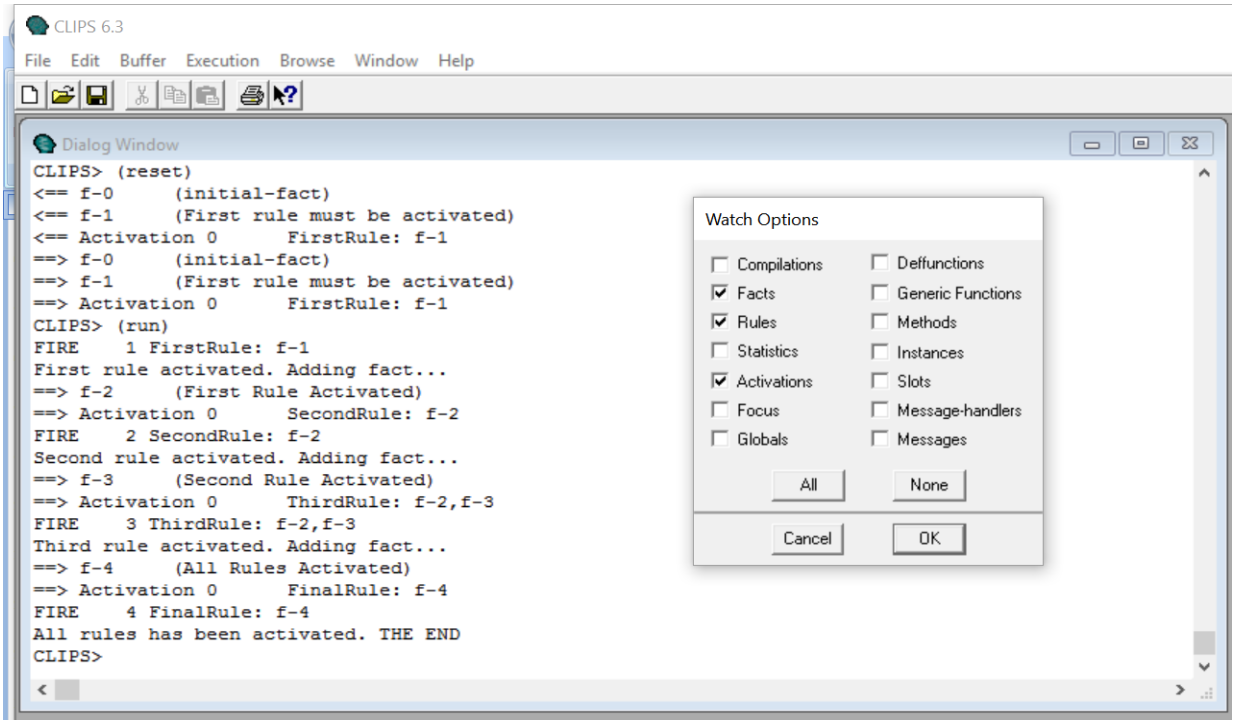

Рис. 4.5. Діалогове вікно з результатами виконання команди *run* та відстеження введення й виведення з пам’яті фактів і виконання правил, а також відповідні налаштування вікна Watch Options.

Для повторного запуску правила можна застосувати команду *refresh*. При виконанні цієї команди (синтаксис команди – (refresh <rule-name>)) в робочий список правил поміщається активоване правило з тих, запуск яких вже відбувся раніше при виконанні команди *run*, а ім’я якого задане в *<rule-name>* (з урахуванням умови, що факти, що викликали його активування, все ще присутні в списку фактів).

У Windows-версії CLIPS цю операцію можна здійснити за допомогою відповідної опції випадаючого вікна Defrule Manager підменю Browse. У вікні Defrule Manager показані всі правила, які були активовані та виконані командою *run* та опції, які дозволяють ними оперувати. Для відновлення правила воно від-

мічається у списку вікна Defrule Manager та натискається кнопка Refresh. Після цього у діалоговому вікні CLIPS буде виведено відповідну команду на виконання відновлення вказаного правила (рис. 4.6).

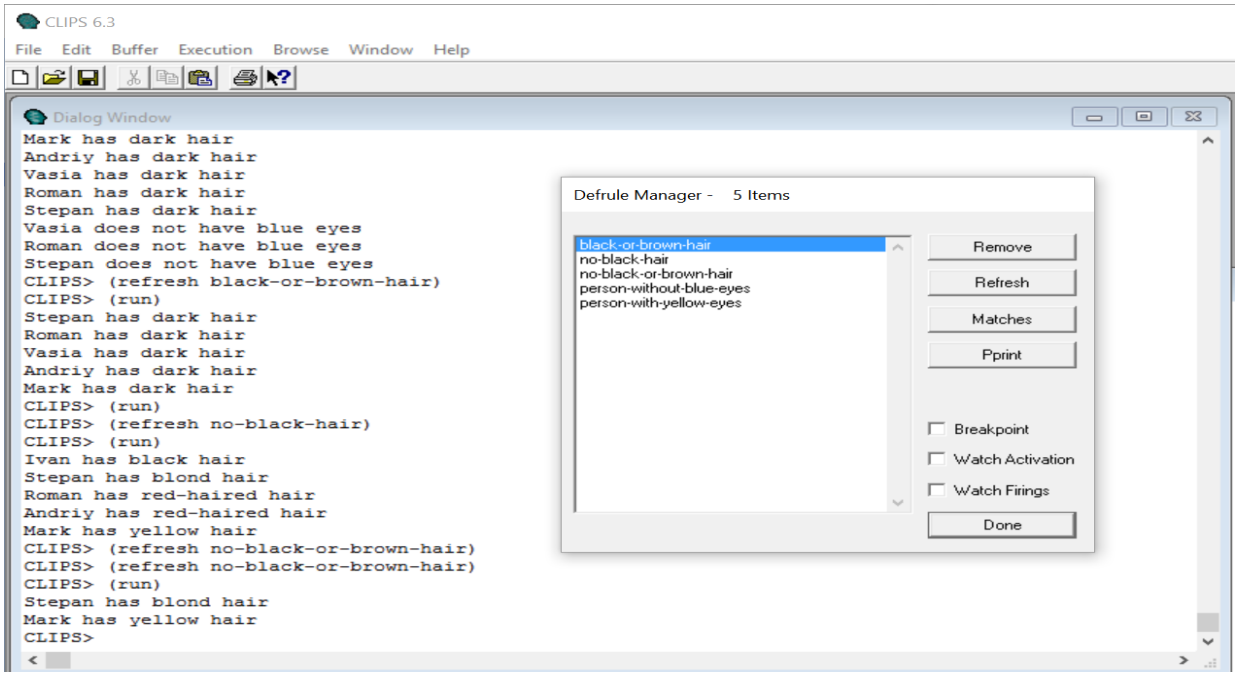

Рис. 4.6. Результати виконання команди *refresh* у діалоговому вікні CLIPS та вікно Defrule Manager.

Після відновлення правила можна знову запустити його на виконання командою *run*.

Порядок виконання роботи.

1. Засвойте особливості створення правил у CLIPS, їх структуру та особливості синтаксису конструктора *defrule*.
2. Сформулюйте правила на природній мові та потрібні для їх запуску впорядковані і невпорядковані факти;
3. Запустить CLIPS, встановіть потрібні умови трасування;
4. Зайдіть у вбудований редактор і наберіть у ньому текст коду кількох правил, які передбачають їх активування шляхом введення різних впорядкованих та

невпорядкованих фактів, та введіть її в CLIPS.

5. Почергово введіть кожен факт за допомогою команди *assert*, і, ввівши команду *run*, переконайтеся у виконанні відповідного правила.
6. Написати код програми послідовного виконання чотирьох правил, в якій передбачити:

- введення, за допомогою конструктора *deffacts*, факту, що задовольняє умову запуску 1-го правила;
- використовуючи конструктор *defrule*, записати чотири правила, супроводжуючи кожну частину правила коментарем, причому:
  - умовою активування першого правила має бути факт, введений за допомогою конструктора *deffacts;*
  - результатом виконання кожного правила має бути введення в список факту про його активування;
  - передбачити виведення на екран повідомлення про активування правила;
  - в умовній частині кожного правила, починаючи з другого, повинен бути факт активування попереднього правила;
  - активування останнього правила має закінчитися виведенням на екран повідомлення, що всі правила активовані.

2. Запишіть набраний код в файл lab4-rules.
3. Замаркуйте написаний код і натисніть комбінацію правил CTRL-M (або натисніть кнопку Batch Selection в меню Buffer).
4. Перевірте правильність написання коду програми. Якщо код введено без помилок, в командному рядку з’явиться запрошення CLIPS>; якщо додавання правил відбулося з помилкою (про що з’явиться відповідне повідомлення з причиною помилки), уважно перевірте синтаксис написання коду відповідного правила.
5. Використовуючи команду *facts* (або опцію Deffacts Manager в меню Browse), перевірте наявність введення в CLIPS потрібного факту.
6. Використовуючи команду *rules* (або Defrule Manager в меню Browse), перевір-

те наявність введення в CLIPS всіх чотирьох правил.

7. Запустіть виконання програми, послідовно запустивши команди *reset* і *run*. Пересвідчіться у виконанні всіх правил програми.
8. Очистіть пам’ять середовища за допомогою команди *clear*.
9. Переконайтесь, що зміна порядку введення в CLIPS коду окремих правил не вплине на хід виконання програми. Для цього:
   - у вбудованому редакторі змініть у коді програми порядок введення правил.
   - знову завантажте змінений код програми і виконайте процедури 3-7.
10. Модифікуйте перше правило для запуску його без введення початкового факту.
11. Видаліть початковий факт з коду програми.
12. Завантажте модифікований код написаної програми в CLIPS та перевірте її виконання.
13. Запишіть результати проведеної роботи (вміст діалогового вікна) у текстовий файл.
14. Використовуючи опцію *print*, роздрукуйте результати роботи.
15. Напишіть звіт про виконану роботу; при оформленні звіту використовувати скріншоти окремих етапів роботи.
16. В додатку до звіту представити код написаної програми.

Контрольні запитання:

1. Що таке правила і яку роль вони відіграють у продукційних експертних системах?
2. Яка структура правила-продукції?
3. Для чого потрібна ліва частина правила і як її ще називають?
4. Які функції виконує права частина правила?
5. Що являє собою конструктор *defrule* і для чого він потрібний?
6. Який синтаксис правила в CLIPS?
7. Яка різниця між обов’язковими та необов’язковими елементами правила?

8. Що таке ім’я правила, як воно записується і для чого потрібне?
9. Що таке активування правила і як воно здійснюється?
10. Що потрібно для забезпечення активування правила?
11. Яким чином формується список активованих правил?
12. Що таке *agenda*?
13. Як можна запустити правило у CLIPS?
14. Які дії виконує вирішував після введення команди *run*?
15. Чи можна знову запустити виконання правила в CLIPS після того, як воно вже виконане?
16. Які команди можна використати для повторного запуску правила в CLIPS?
17. Які функції команди *refresh* і як її можна виконати, використовуючи графічний інтерфейс?
18. Як запускається виконання програми в CLIPS?
19. Чи можна запустити правило, в якому не задані умови його виконання? Якщо так – що для цього потрібно?
20. Чи буде виконане правило, ліва частина якого «пуста» – тобто не містить ніяких умов?
21. Чи впливає порядок введення правил у середовище CLIPS на черговість їх застосування вирішувачем?
22. Як можна переглянути код введеного в список правил CLIPS правила?
23. Як за допомогою графічного інтерфейсу переглянути список правил, введених у CLIPS?
24. Що таке трасування і для чого воно потрібне?
25. Коли виникає конфлікт правил і як цей конфлікт вирішується?
26. Що таке пріоритет правила і яким чином він встановлюється в CLIPS?
27. Що таке стратегія вирішення конфліктів і в чому її суть?
28. Які стратегії вирішення конфліктів застосовуються в CLIPS?
29. Як можна встановити потрібну стратегію вирішення конфлікту в CLIPS?

Література:

1. CLIPS. A Tool for Building Expert Systems [Електронний ресурс], 2016. – Режим доступу: https://sourceforge.net/projects/clipsrules/files/CLIPS/6.30/clips_documentation_630.zip/download
2. CLIPS Rule Based Programming Language [Електронний ресурс], 2016. – Режим доступу: https://sourceforge.net/projects/clipsrules/files/CLIPS
3. CLIPS Reference Manual. Volume I. Basic Programming Guide. Version 6.30 March 17th 2015 – 416 p. [Електронний ресурс] – Режим доступу: http://clipsrules.sourceforge.net/documentation/v630/bpg.pdf
4. Джарратано Джозеф. Экспертные системы. Принципы разработки и программирование, 4е изд: Пер. с англ. / Джозеф Джарратано, Гари Райли. – М., Изд. дом “Вильямс”, 2007. – 1152 с.
5. Частиков А. П. Разработка экспертных систем. Среда CLIPS. / А. П. Частиков., Т. А. Гаврилова, Д. Л. Белов. – СПб., «БХВ-Петербург», 2003. – 608 с.

Лабораторна робота 5.

Тема: ВИКОРИСТАННЯ ЗМІННИХ ТА ОБМЕЖЕНЬ ПОЛІВ У ЛІВІЙ ЧАСТИНІ ПРАВИЛ. СТВОРЕННЯ ПРОГРАМИ «ПІДСУМКИ ЕКЗАМЕНАЦІЙНОЇ СЕСІЇ»

Мета роботи: Здобути навички написання програми, яка використовує правила з різними обмеженнями полів в LHS, в середовищі CLIPS.

Завдання роботи:

Освоїти особливості використання змінних та обмежень полів у лівій частині правил. Використовуючи змінні та умовні елементи, а також пріоритетність правил, написати код програми “Результати екзаменаційної сесії». Створити варіанти програм з використанням впорядкованих і невпорядкованих фактів.

Вихідні матеріали:

5. Особливості і засоби створення умовної частини правил в CLIPS.

Правила CLIPS можна порівняти з операторами типу *if-then* процедурних мов програмування. Проте є і суттєві відмінності у їх застосуванні. Зокрема, умова оператора *if* при виконанні зв’язки операторів *if-then* в програмі, написаній на процедурній мові програмування, перевіряється лише тоді, коли програма передає йому управління. З правилами CLIPS ситуація кардинально інша. Як уже згадувалося вище, блок логічного виведення системи під час своєї роботи постійно відстежує всі правила, умови яких виконуються, і, таким чином, будь-яке з введених в систему правил може бути активоване у будь-який момент роботи, як тільки воно стає застосовним (тобто, коли виконується його умовна частина з появою відповідних фактів в робочій пам’яті).

Як уже згадувалося вище, у найпростішому випадку правила являють собою порівняно нескладні структури з єдиною умовою в їх лівій частині, яку необхідно задовольнити для їх запуску, і які ілюструють прості засоби зіставлення шаблонів з фактами. Однак, такі правила мають досить обмежений ресурс використання, оскільки для виконання складніших запитів у цьому випадку потрібно використати значну їх кількість. У більшості випадків при розробці експертних систем виникає необхідність у створенні більш потужних програм, в яких для вирішення поставленої задачі використовуються як складніші правила, так і їх групи. Зокрема, досить часто виникає необхідність у створенні таких правил, умовна частина яких містила б низку елементів, які не тільки можуть бути подібними, але і протилежними за своєю суттю, навіть взаємно виключними.

Для створення таких правил використовується арсенал засобів, представлених в CLIPS, серед яких чільне місце займає використання обмежень полів та використання різних типів умовних елементів. Їх застосування дозволяє використовувати одне правило для виконання функцій кількох простих правил і, до того ж, надає можливість виконувати їх узгодження не тільки з існуючими, але і з ще не існуючими на поточний момент фактами.

5.1. Ліва частини правила.

Після свого запуску CLIPS робить спроби зіставити шаблони лівої частини кожного правила з фактами зі списку фактів. Якщо всі шаблони LHS правила узгоджуються з наявними фактами, то це правило вважається відповідним поточній ситуації предметної області задачі, активується і поміщається в робочий список правил – список активованих правил, який під час роботи CLIPS може бути або порожнім (умови активування для жодного правила з бази правил не виконані), або ж в ньому може знаходитися будь-яка кількість правил з бази знань, ліві частини яких задовольняються наявними в пам’яті системи фактами.

Якщо в лівій частині правила не вказаний жоден умовний елемент, CLIPS

автоматично підставляє *initial-fact* як його умову-шаблон. Таким чином, таке правило активується всякий раз при появі в базі знань факту *initial-fact*, у т. ч. при виконанні команди *reset*.

Передумова (або ліва частина правила) правила задається за допомогою набору умовних елементів, який зазвичай складається з низки умов, застосовних до деяких зразків. Заданий набір зразків правила використовується системою для зіставлення із наявними у пам'яті CLIPS фактами з метою визначення його відповідності умовам наочної області задачі на певний момент і, фактично, визначає його застосовність. Тому формуванню лівої частини правила в CLIPS приділяється виняткова увага.

Типи умовних елементів лівої частини правила

Загалом, для формування лівої частини правил у CLIPS використовують вісім типів умовних елементів:

CE-зразок – умовний елемент, що найчастіше використовується у правилах – містить обмеження, які служать для визначення, чи задовольняє який-небудь факт із списку фактів заданому зразку;

test CE – умова, яка використовується для оцінки виразу як частини процесу зіставлення зразків;

and CE – умова, яка застосовується для визначення групи умов, кожна з яких повинна бути задоволеною;

or CE – умова, яка застосовується для визначення однієї умови з деякої їх групи, яка повинна бути задоволена;

not CE – умова, яка застосовується для визначення умови, яка не повинна бути задоволена;

exists CE – умова, яка застосовується для перевірки наявності, принаймні, одного збігу факту (або об'єкту) з деяким заданим зразком;

forall CE – умова, яка застосовується для перевірки виконання деякої умови для всіх заданих умовних елементів правила;

logical CE – умова, яка дозволяє виконати додавання фактів і створення об'єктів в правій частині правила, пов'язаних з фактами і об'єктами, що співпали із заданим зразком в лівій часта правила (підтримка достовірності фактів в базі знань).

Кожен з вказаних умовних елементів має свій синтаксис і межі застосування.

Зразок (pattern).

Зразки – умовні елементи, що найчастіше використовуються в правилах – складаються з набору обмежень, які використовуються для опису того, які факти повинні задовольняти умови, що визначаються зразком. Серед обмежень, які найчастіше використовуються для формування зразків, розрізняють *обмеження полів*, *групові символи* і *змінні.*

О*бмеження полів* використовуються для перевірки простих полів або слотів об'єктів і можуть складатися тільки з одного *символьного обмеження*; проте, декілька обмежень можна сполучати разом. Окрім символьних обмежень, CLIPS підтримує три інших типи обмежень: *об'єднуючі обмеження, предикатні обмеження* й *обмеження, що повертають значення.*

Групові символи використовуються при зіставленні зразків у ситуації, коли просте поле або група полів можуть приймати будь-які значення.

Змінні застосовуються для зберігання значення поля, що може бути згодом використане в цьому ж правилі для формування іншого умовного елемента його лівої частини або ж як аргумент дії у правій частині.

Як уже згадувалося, перше поле будь-якого зразка обов'язково повинне бути значенням типу *symbol* і повинне відповідати імені шаблона, створеного явно за допомогою конструктора *deftemplate* або неявно (тобто при введенні впорядкованого факту); воно використовується CLIPS для визначення, чим є даний зразок – фактом (упорядкованим чи шаблонним) чи об'єктом (у цьому випадку використовується ключове слово *object*).

Для завдання імен слотів невпорядкованих фактів також повинні використовуватися значення типу *symbol*. У слотах простих полів зразків може використовуватися тільки одне обмеження поля; також тут не можуть бути присутні групові символи або змінні. У складених слотах може використовуватися будь-яка кількість обмежень поля.

Символьні обмеження.

Символьними називаються обмеження, що визначають точну відповідність між полями факту і зразком правила.

Символьні обмеження визначають точну відповідність між полями факту і зразком і складаються з констант, значень типу *symbol*, рядків або імен об'єктів. Всі символьні обмеження при зіставленні зразків з фактами повинні точно збігатися за всіма значеннями полів, інакше факт не буде вважатися таким, що задовольняє ці зразки. Іншими словами, правило, в лівій частині якого є символьне обмеження, буде активоване тоді і лише тоді, коли його ліва частина повністю співпадає з фактом, який є в робочій пам'яті системи. Такі обмеження мають синтаксис

(<обмеження-1> ... <обмеження-n>)

для впорядкованих фактів і

(<ім'я-шаблону >
(<ім'я-слота-1><обмеження-1>)
…
(<ім'я-слота-n><обмеження-n>))

– для невпорядкованих.

Наприклад, правило пошуку за адресою, записане з використанням символьних обмежень в його LHS як

```
(defrule find-by-address
(address Lviv gen. Tarnavskogo str 107 room 217)
=>
(printout t crlf “Laboratory of gamma-ray spectrometry” crlf)
(printout t crlf “Address: Lviv gen. Tarnavskogo str 107 room 217” crlf)),
```

буде активоване, якщо в пам’яті системи буде присутній факт

(address Lviv gen. Tarnavskogo str 107 room 217),

який повністю збігається з відповідним символьним обмеженням зразка лівої частини правила, і після його виконання на монітор буде виведено напис:

Laboratory of gamma-ray spectroscopy
Address: Lviv gen. Tarnavskogo str 107 room 217

Результати послідовного введення і виконання вказаного правила та відповідного впорядкованого факту CLIPS показане на рис. 5.1.

Однак, використання лише символьних обмежень зумовлює нагромадження в базі знань експертної системи значної кількості правил, особливо тоді, коли логіка її побудови вимагає, щоб консеквентна частина правила могла бути реалізована при наявності в робочій пам’яті кількох різних фактів. Інакше кажучи, при розробці бази правил ЕС може виникнути ситуація, коли одна і та ж дія правила може бути зумовлена різними причинами (фактами) або їх частинами. Для того, щоб забезпечити активування правила з однією консеквентною частиною, але за умови задоволення його антецедентної частини кількома різними фактами (або їх частинами), а також більш гнучких можливостей оперуванням значеннями полів як в лівій, так і в правій частині правил в CLIPS передбачена можливість використання при зіставленні полів у зразках змінних та групових символів, а також різних типів обмежень полів.

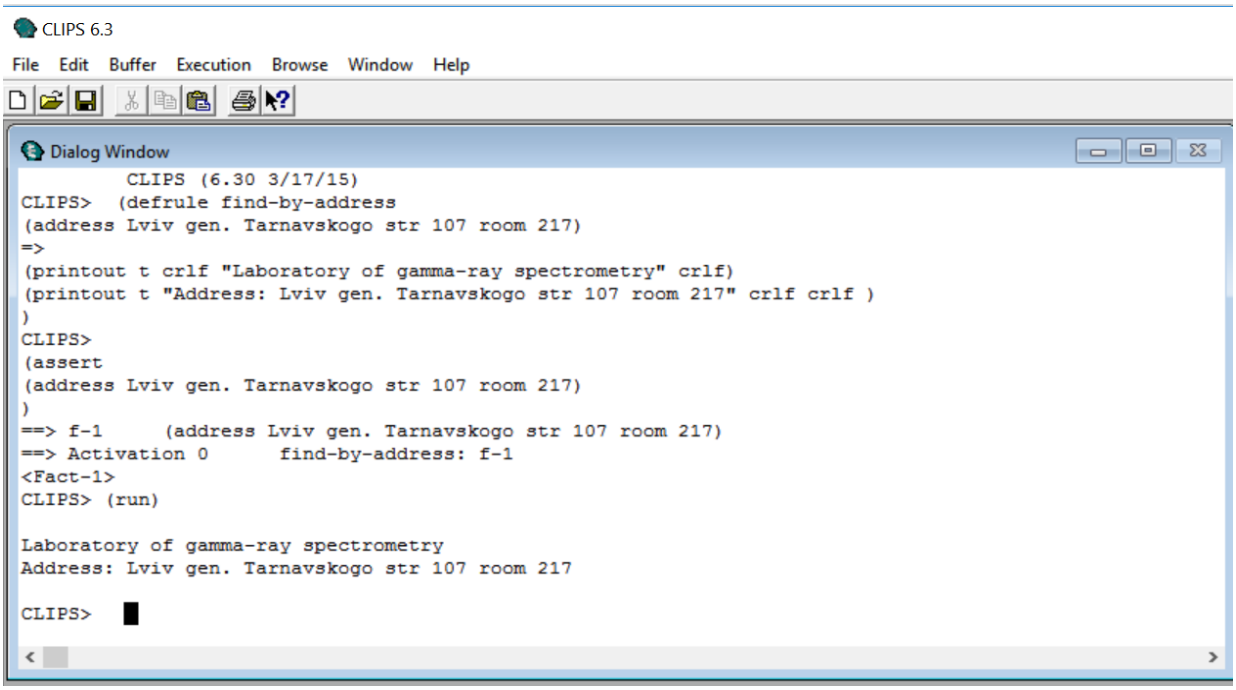

Рис. 5.1. Результат виконання CLIPS правила *find-by-address*.

5.2. Змінні та групові символи у CLIPS.

Як і в інших мовах програмування, в CLIPS для зберігання значень використовуються змінні, які, згідно особливостей свого використання, можуть бути локальними (чинними тільки в межах певного модуля, в якому вони визначені) або глобальними (які чинні у всьому середовищі CLIPS. Спільним для обох видів змінних є те, що в своєму синтаксисі вони обов’язково повинні використовувати знак «?».

5.2.1 Локальні змінні.

Локальні змінні CLIPS записуються як:

?<variable-name>

Ідентифікатор змінної завжди починається зі знака «?», за яким (без пробілу) слідує її ім’я (ім’я поля або слота факту) типу <symbol>.

Приклади змінних: ?x, ?x1, ?name, ?noun, ?color.

Усі змінні, окрім глобальних, вважаються локальними і можуть використовуватися тільки в рамках опису текучої конструкції (модуля); до локальних змінних можна звертатися тільки всередині опису, але вони не визначені поза ним.

Перед використанням змінної їй необхідно привласнити значення.

Зазвичай змінні в CLIPS описуються і отримують свої значення в лівій частині правила. Отримавши значення, змінна зберігає його незмінним при використанні як в лівій, так і в правій частині правила, якщо тільки це значення не змінюється в правій частині за допомогою функції *bind*.

Крім значення самого факту, змінній також може бути присвоєне і значення адреси факту. Це може виявитися зручним при необхідності маніпулювати фактами безпосередньо з правила – наприклад, для того, щоб визначити, який факт буде змінюватися, необхідно привласнити змінній адресу конкретного факту. Для такого присвоєння використовується комбінація знаків «<-». Присвоєння адрес відбувається в лівій частині правила з синтаксисом:

<pattern-address> :: = ?<name-variable> <- <pattern>;

отримане значення називається адресою зразка (pattern-address).

Стрілка «<-» – необхідна частина синтаксису присвоєння. Змінна, пов’язана з адресою факту або об’єкта, може порівнюватися з іншою змінною або використовуватися зовнішньої функцією. Змінна, пов’язана з адресою факту, також може бути також використана для подальшого обмеження полів в зразку умовного виразу. Однак, потрібно пам’ятати, що не можна пов’язувати змінну в умовному елементі *not*.

Після зв’язування цю зміну можна використовувати в командах правої частини правила – *retract*, *modify* або *duplicate* замість індексу факту.

5.2.2. Глобальні змінні.

Для визначення глобальних змінних, які, на відміну від локальних, чинні всюди (тобто у будь-якому модулі програми середовища CLIPS), використовуєть-

ся конструкція *defglobal* з синтаксисом:

(defglobal [<defmodule-name>] <global-assignment>*).

У цьому визначенні параметр <global-assignment> задається наступним чином:

<global-assignment> ::= <global-variable> = <expression>,

а параметр <global-variable> визначається як:

<global-variable> ::= ?*<symbol>*

Необов'язковий терм <defmodule-name> представляє собою модуль, в якому повинні бути визначені глобальні змінні – якщо цей параметр не заданий, то глобальні змінні поміщаються в поточний модуль.

Імена глобальних змінних починаються і закінчуються знаком «*», завдяки чому можна легко відрізнити локальну змінну, що позначається як ?*x*, від глобальної, яка позначається як ?**x**. Як видно з визначення конструкції *defglobal,* початкове значення для кожної глобальної змінної задається шляхом обчислення виразу <expression> і присвоювання отриманого значення змінній *defglobal*.

До глобальної змінної можна звернутися в будь-якому місці і в будь-якій частині будь-якого правила; її значення залишається незалежним від інших конструкцій. Глобальні змінні CLIPS подібні до глобальних змінних у процедурних мовах програмування – з тією відмінністю, що на них не накладається обмеження на зберігання даних тільки одного типу.

Глобальні змінні можуть використовуватися в будь-якому місці, де може використовуватися локальна змінна; вони не можуть використовуватися як змінний параметр для конструкторів *deffunction, defmethod* або в оброблювачі повідомлень.

Маніпулювати конструкціями *defglobal* (вивести на зовнішній пристрій її текстове представлення, видалити, відобразити список, імена і значення глобальних змінних) можна за допомогою команд з синтаксисом:

(get-defglobal-list [<module-name>])

(list-defglobals [<module-name>])

(show-defglobals [<module-name>])

(ppdefglobal <defglobal-name>)

(undefglobal <defglobal-name>)

Функція *get-defglobal-list* повертає багатозначне значення, яке містить список конструкцій *defglobal*.

Команда *list-defglobals* призначена для відображення в діалоговому вікні списку імен всіх визначених у системі глобальних змінних. Якщо необов'язковий параметр <module-name> не вказаний, то дана команда виводить імена глобальних змінних, визначених у поточному модулі. Якщо ж він містить ім'я конкретного модуля, команда *list-defglobal* виводить список змінних, визначених у заданому модулі. У випадку використання як параметра символу «*» команда виведе в діалогове вікно імена всіх глобальних змінних, визначених у всіх модулях системи.

Команда *show-defglobals*, на відміну від команди *list-defglobals*, виводить у діалогове вікно CLIPS не тільки імена глобальних змінних, але і їх значення. В іншому ці дві команди практично ідентичні.

Потрібно пам'ятати, що як <defglobal-name> для команд *ppdefglobal* і *undefglobal* повинно використовуватись ім'я глобальної змінної без початкових і кінцевих знаків «*» (напр., *x*, а не **x**).

Приклад визначення глобальних змінних та застосування до них вказаних функцій показані на рис. 52.

Параметр <value>*, що задає вираз для значення нової змінної (локальної чи глобальної), загалом є необов'язковим. У випадку глобальної змінної, коли він не заданий, для неї буде встановлено початкове значення, задане в конструкторі *defglobal*. У випадку, якщо вираз було задано, то його значення буде обчислене і результат присвоєний змінній. Якщо було задано декілька виразів, всі вони будуть

обчислені, а з їх результатів буде створене складене поле, яке буде присвоєне для глобальної змінної.

Функція *bind* повертає значення FALSE в разі, якщо змінній з якоїсь причини не було присвоєно ніякого значення. В іншому випадку функція повертає значення, присвоєне змінній (розраховане згідно виразу, який визначає це значення).

Оскільки змінні в CLIPS слабо типовані, типи значень, що присвоюються одній і тій же змінній під час роботи програми, у різні моменти часу можуть не збігатися.

Команда *reset* повертає всім глобальним змінним початкові значення, визначені при їх оголошенні конструктором *defglobal*.

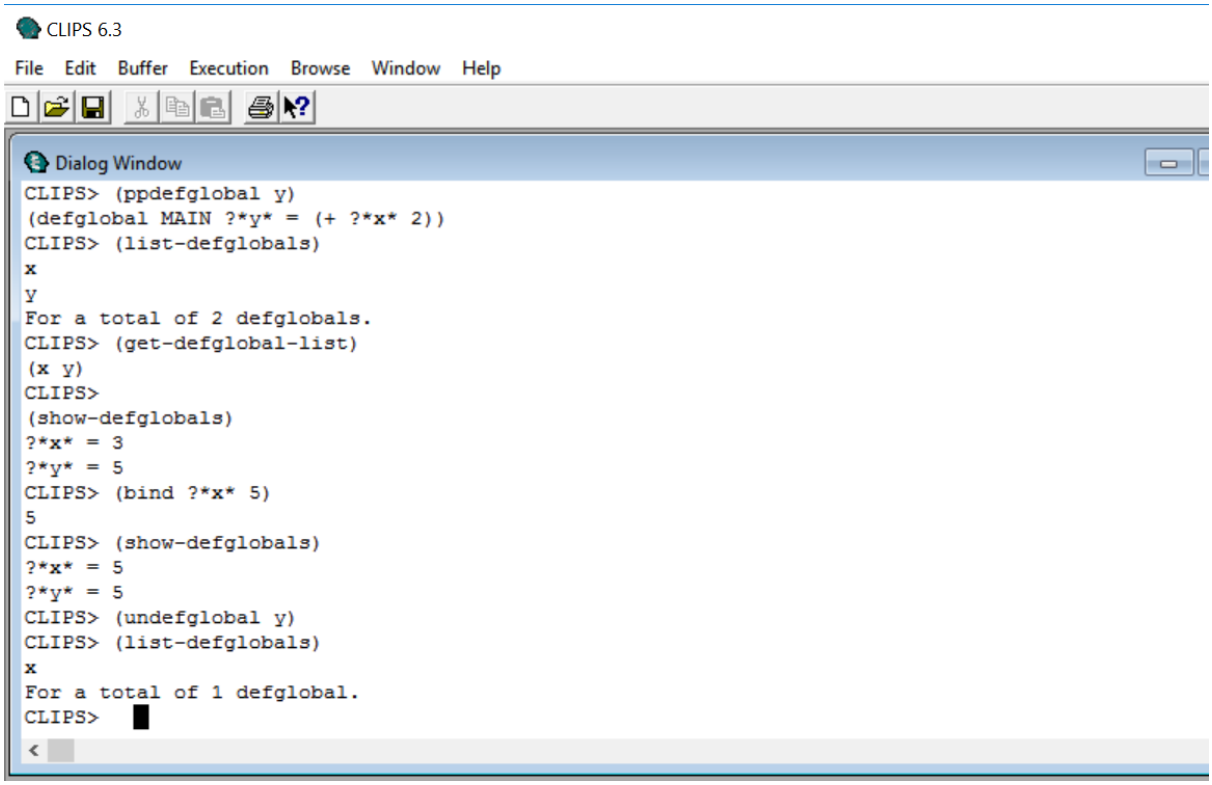

Рис. 5.2. Визначення глобальних змінних та результати застосування функцій для маніпуляцій з ними.

5.2.3. Використання групових символів.

Для реалізації зіставлення полів в зразках у CLIPS використовуються два різні типи групових символів – для простих і складених полів, відповідно, які інте-

рпретуються як місце для підстановки деяких частин фактів, заміняють будь-які поля зразка й можуть приймати які завгодно значення цих полів.

Груповий символ для простого поля записується за допомогою знаку «?», який відповідає одному будь-якому значенню, збереженому в заданому полі, а для складеного поля – за допомогою комбінації знаків «$?» і відповідає в цьому випадку послідовності значень, збереженій в складеному полі. Групові символи для простих і складених полів можуть комбінуватися в будь-якій послідовності. Не допускається використання групового символу складеного поля для простих полів.

Використання групових символів покажемо на прикладі. Нехай у CLIPS уведені шаблон *deftemplate car* та невпорядковані факти:

```
(car (color black) (model Ford) (owner Petriv Sergiy))
(car (color white) (model Opel) (owner Ivaniv Petro)),
```

а також правило пошуку власника чорного автомобіля:

```
(defrule owner-of-a-black-car
 (car
(color black)
(model ?y)
(owner $?x) )
=>
(printout t $?x « is a owner of a black car « ?y crlf) )
```

В результаті виконання цього правила отримаємо результат (рис. 5.3):

(Petriv Sergiy) is a owner of a black car Ford

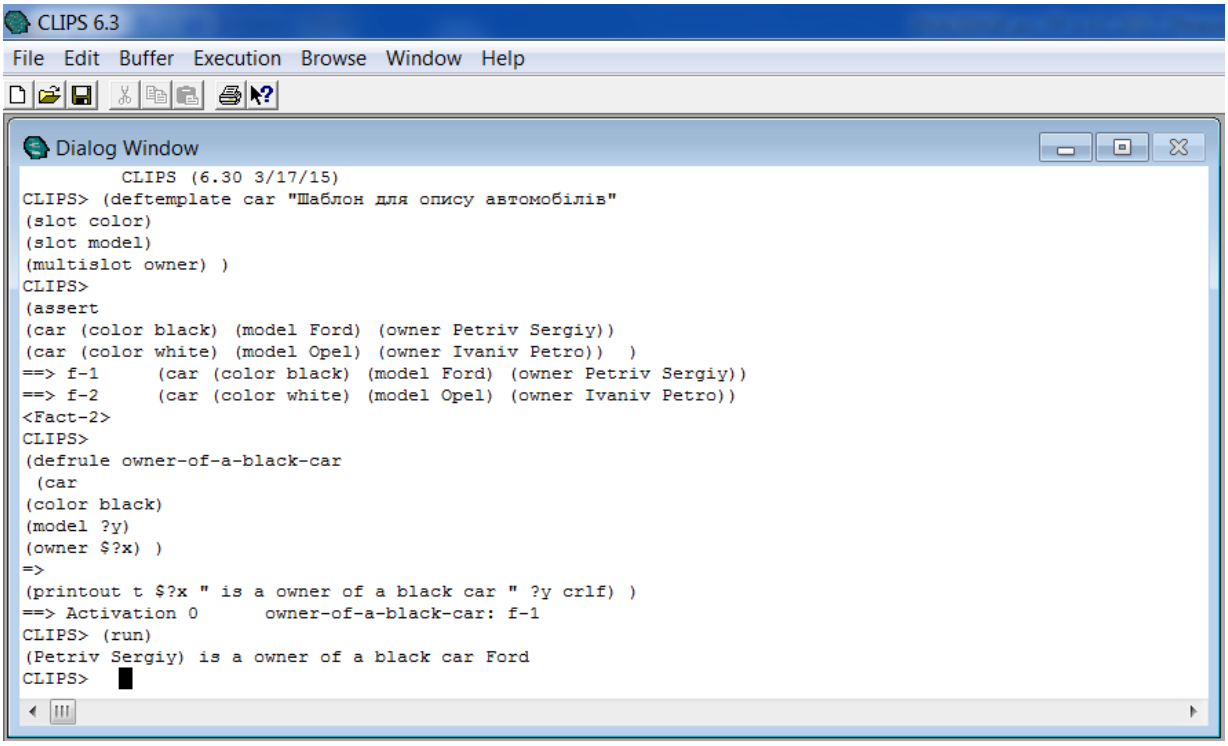

Рис. 1.3. Результати виконання правила owner-of-a-black-car з груповими змінними.

5.3. Обмеження полів та особливості їх використання.

Обмеження значень полів: єднальні (зв'язуючі) обмеження

Використання обмежень на значення полів дозволяє обмежити значення, що приймаються зразками в лівій частині правила і, відповідно надати більше можливостей при їх створенні.

В CLIPS розрізняють три види обмежень на значення полів, які мають значення логічнного «НІ», логічного «АБО», логічного «І» й позначаються при написанні коду як «~», «|» та «&», відповідно; їх ще називають *єднальними обмеженнями*. Синтаксис єднальних обмежень:

~ <елемент>

<елемент-1>| <елемент-2> ... | елемент-n>

<елемент-1>&<елемент-2> ... & елемент-n>

Тут <елемент> може бути змінною, пов'язаною із простим або складовим полем, обмеженням або зв'язаним обмеженням.

Обмеження першого типу (логічне «НІ», «~») діє на значення, яке слідує безпосередньо за ним, і говорить про те, що дане поле не може приймати це значення. Наприклад:

```
(defrule walk
        (light-color ~green)
=>
        (printout t «Do not walk» crlf)).
```

Використання обмеження «~» (логічне «НІ») в лівій частині цього правила показує, що для задоволення умови правила це поле може приймати будь-яке значення кольору, але не *green*. Інакше кажучи, обмеження «~» задовольняється тоді, якщо вказане наступне за ним значення поля не задовольняється.

Обмеження другого типу «|» (логічне «АБО») вказує на те, що поле може приймати будь-яке значення з тих, які об’єднані цим обмеженням. Наприклад:

```
(defrule cautious
        (light-color  yellow | blinking-yellow)
=>
        (printout t «Be cautious.» crlf)).
```

Використання обмеження «|» показує, що це правило активуватиметься тоді, коли поле матиме одне з вказаних значень – *yellow* або *blinking-yellow*.

Обмеження третього типу (логічне «І», «&») вказує на те, що повинні задовольнятися обидва об’єднаних за його допомогою обмеження. Відсутність будь-яких обмежень свідчить, що поля лівої частини правила об’єднуються саме логічним “І”.

Зазвичай, обмеження & використовується тільки для об’єднання з іншими обмеженнями або зв’язування змінних.

Єднальні обмеження можуть використати і вже зв'язані змінні, й самі зв'язувати змінну зі значенням якогось поля. Якщо ім'я змінної зустрілося вперше, то для обмеження будуть використовуватися інші члени умовного елемента, а змінна буде пов'язана з відповідним значенням поля. Якщо ж змінна вже була зв'язана, то її значення працює як додаткове обмеження для даного поля.

Єднальні обмеження можуть комбінуватися в правилах майже довільним чином і у без обмежень їх кількості. Найвищий пріоритет серед них має обмеження "~", далі слідують "&" й "|". У випадку однакового пріоритету обмеження обчислюється зліва направо. Існує одне виключення із правил пріоритету, що застосовується при зв'язуванні змінних: якщо перше обмеження – це змінна й за нею слідує &, то змінна є окремим обмеженням.

Особливості використання обмежень усіх трьох типів розглянемо на прикладі. Введемо в CLIPS таке правило:

```
(defrule cautious
        (light-color ?color & ~red & ~yellow & ~blinking-yellow)
  =>
        (printout t «You can walk: color « ?color crlf))
```

Виконання вказаних обмежень свідчить, що правило спрацює лише тоді, коли значенням поля не буде *red*, *yellow* чи *blinking-yellow*. Якщо ми тепер введемо в систему факти:

```
(light-color red)
(light-color yellow)
(light-color blinking-yellow)
(light-color green)
```

то правило cautious буде активоване лише фактом (light-color green) (і за відсутно-

сті перших трьох фактів), а в діалоговій стрічці CLIPS буде видано результат (рис. 1.4):

You can walk: color green.

Потрібно відмітити, що наше правило спрацює при появі у списку факту, який міститиме будь-який колір, окрім згаданих в його умовній частині.

Предикатні обмеження

Іноді необхідно обмежити поле, ґрунтуючись на істинності деякого логічного виразу. Для цього CLIPS використовує предикатні обмеження, які дозволяють викликати т. з. предикатні функції (під такими розуміють функції, які повертають значення TRUE при відповідності фактів зі списку заданим умовам і FALSE при їх невідповідності) під час процесу зіставлення зразків. Відповідно, якщо предикатна функція повертає значення TRUE, предикатне обмеження задовольняється, а якщо значення FALSE – ні.

Предикатні обмеження записуються за допомогою двокрапки і наступного за ним виклику відповідної предикатної функції:

:<function>

Зазвичай предикатні обмеження використовуються спільно зі зв’язуючими обмеженнями і при зв’язуванні змінних (тобто якщо є змінна, яку потрібно зв’язати з деяким полем і потрібно одночасно її протестувати, то це можна зробити об’єднанням її з предикатним обмеженням).

CLIPS надає низку готових предикатних функцій (ознайомитися з властивостями і синтаксисом яких можна, напр., в документації на CLIPS; однак, користувач і сам може створити потрібну предикатну функцію, використавши конструктор *deffunction.*

Використання предикатних обмежень розглянемо на прикладі. Ведемо в CLIPS список фактів, що містить імена людей, які мають різні дні народження:

(deffacts birthday

```
(birthday Ivan 1998 January 14)
(birthday Petro 1995 January 17)
(birthday Andriy 1997 January 27)
(birthday Stepan 1998 Aptil 17)
(birthday Sonia 1999 March 7)
(birthday Sofia 1998 January 17)
(birthday Varia 1998 January 20))
```

Нехай потрібно знайти людей, які народилися не в 1998 р. і дата народження яких не є 17 днем місяця. Для цього можна використати правило, яке містить предикатні обмеження:

```
(defrule Find-data-birthday1
(birthday ?x ?y& :(<> ?y 1998) ?z ?z1&:(integerp ?z1)&:(<> ?z1 17))
=>
(printout t ?x « birthday is « ?y «  « ?z «  « ?z1 «.»crlf  )).
```

Ввівши записане вище правило, після застосування команд *reset* та *run* отримаємо результат (рис. 5.5), який показує, що заданим умовам пошуку задовольняє двоє людей з введеного списку:

```
Sonia birthday is 1999  March  7.
Andriy birthday is 1997  January 27
```

Обмеження, які повертають значення

В обмеженнях можливе використання значень, повернутих деякими функціями (у тому числі й зовнішніми). Виклик функції записується за допомогою знака «=« і зазначеної за ним функції:

= <function>

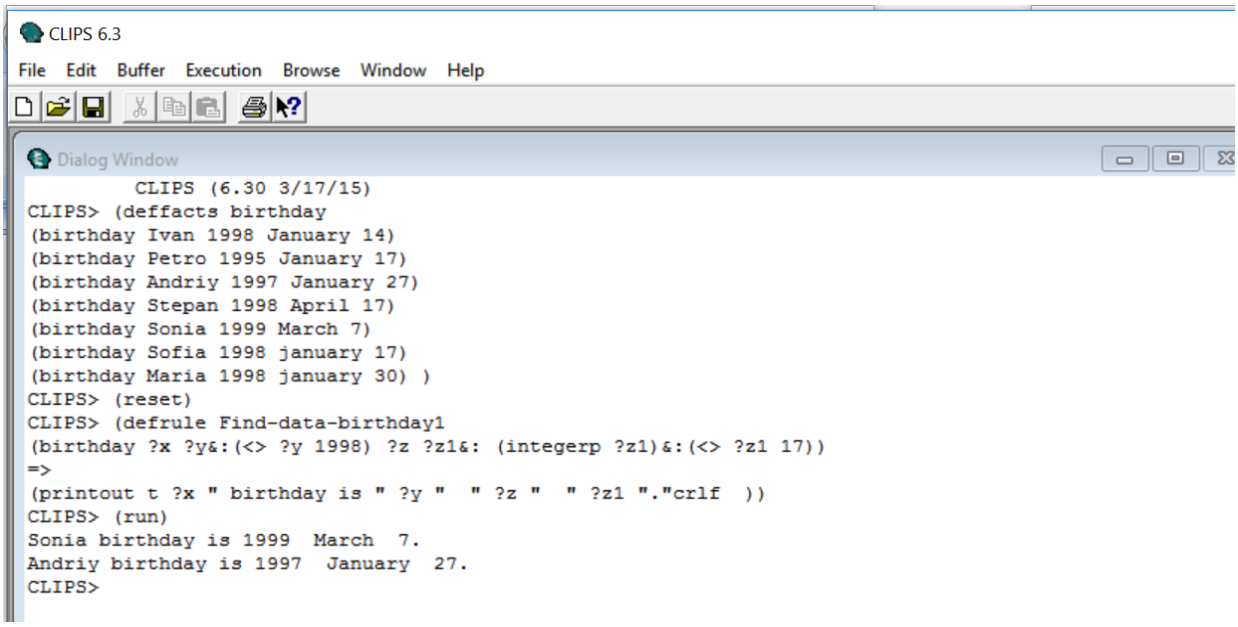

Рис. 5.5. Результат виконання правила Find-data-birthday1.

Значення, що повертається, повинне бути одним з простих типів даних CLIPS. Це значення, повернене функцією, об’єднується зі зразком так, як якби б воно було символьним обмеженням. Потрібно мати на увазі, що функція обмеження обчислюється при кожному зіставленні зразків, а не один раз при визначенні правила.

Розглянемо застосування цього обмеження на прикладі. Нехай є масив даних, введених у CLIPS командою:

```
(assert
(data 1   1 2 3)
(data 2   2 3 8)
(data 3   2 4 6)
(data 4   5 10 15)
(data 5   4 2 6)
)
```

і потрібно визначити рядки у масиві даних, значення членів яких зв’язані певними математичними співвідношеннями – напр., значення другого у два рази більше за

перше, а третій є сумою двох попередніх. Це можна зробити, використавши правило:

```
 (defrule Find-data
(data ?x1 ?x ?y&=(* 2 ?x) ?z&=(+ ?x ?y))
=>
(printout t «row» ?x1 «   « ?x « « ?y « « ?z crlf)).
```

В результаті введення даних та виконання вказаного правила отримаємо, що вказаним умовам задовольняють дані 1, 3 та 4-го рядків масиву:

```
row4   5 10 15
row3   2  4  6
row1   1  2  3
```

Результат вводу вказаного масиву даних та виконання правила Find-data показаний на рис. 5.6.

1.4. Особливості використання умовних елементів в LHS правил

Як уже згадувалося, у CLIPS використовується низка логічних операторів для групування та порядку застосування умовних елементів при формуванні умов виконання правил. Особливості застосування та синтаксис тих з них, що найчастіше застосовуються в правилах, показані нижче.

Умовний елемент *and*.

Визначає, що всі умовні елементи, задані в лівій частині, повинні бути задоволені; на кількість об’єднуваних ним умовних елементів нема ніяких обмежень. Використання оператора *and* означає, що правило буде активоване лише у тому

випадку, якщо всі наявні в лівій частині правила умови задоволені, тобто співпадають з наявними у робочій пам’яті системи фактами. Синтаксис умовної частини правила у цьому випадку:

<умовний-елемент-and> ::= (and< умовний елемент>+).

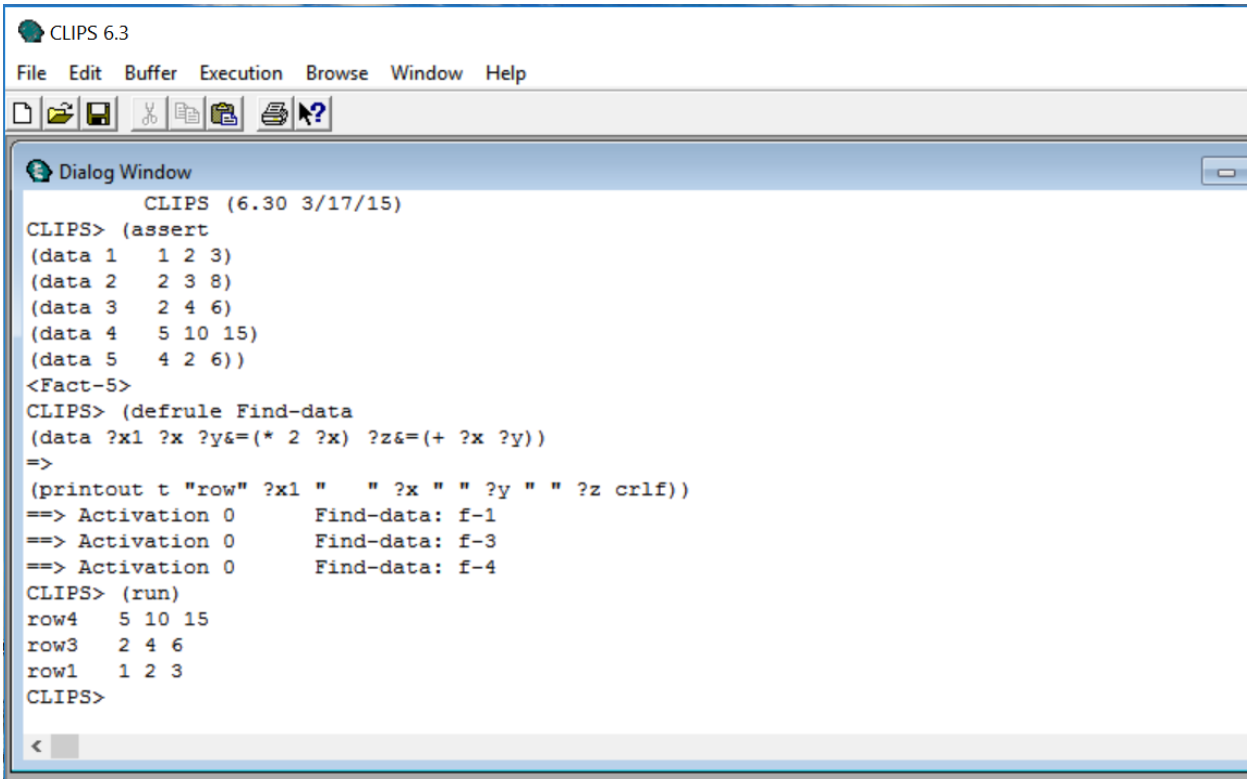

Рис. 5.6. Результат виконання CLIPS правила Find-data.

Умова *and* задовольняється, тільки якщо всі умовні елементи всередині явно заданого ним задоволені.

При відсутності явного задання оператора *and* в умовній частині правила в CLIPS прийнято вважати, що всі умовні елементи в цій частині правила об’єднані неявним умовним елементом *and* і правило знову ж таки активується тоді і тільки тоді, коли всі умови його лівої частини задовольняються.

Умовний елемент *not*.

Іноді виникає необхідність в активуванні правила, виходячи з відсутності

саме якогось конкретного факту в списку фактів – тобто у ситуації, коли необхідно запустити правило, якщо зразок або інший умовний елемент не задовольняється (наприклад, певного факту не існує). Система CLIPS надає таку можливість за допомогою використання умовного елемента *not*, який дозволяє здійснити активування правила, виходячи з відсутності певного конкретного факту в списку фактів; реалізується вона шляхом заперечення певного твердження, яке задовольняється тоді, коли умовний елемент не задовольняється.

Синтаксис умовного елементу *not*:

<умовний-елемент-not> ::= (not <умовний-елемент>)

Потрібно пам'ятати, що один умовний елемент *not* може заперечувати тільки один вираз – для заперечення декількох умовних елементів потрібно застосувати і декілька елементів *not*. В заперечуючому шаблоні можна також використовувати змінні; однак, потрібно мати на увазі, що змінні, вперше пов'язані зі значенням в умовному елементі *not*, зберігають своє значення тільки в області визначення цього ж елемента.

Умовний елемент *or*.

Дозволяє активувати правило задоволенням будь-якого з декількох заданих умовних елементів, які входять в умовну частину правила. Синтаксис елемента:

<умовний-елемент-or> ::= (or <умовний елемент>+).

Оператор *or* може об'єднувати будь-яку кількість умовних елементів правила. Таким чином, якщо будь-який з двох умовних елементів, об'єднаних за допомогою умовного елементу *or*, задовольняється, то і весь вираз вважається задоволеним.

Окрім умовних елементів, що об'єднані умовним елементом *or*, в умовну частину правила можуть входити й інші, які не підлягають умові *or*. Тоді, за визначенням, вважається, що ці елементи разом з тими, що об'єднані елементом *or*, перебувають під дією неявно заданого елемента *and* і, загалом, правило буде активоване тоді, коли, крім виразу під *or*, й решта всіх умовних елементів, що входять

в ліву частину правила (але що не входять в *or*), також буде задоволена. Таким чином, за допомогою явного використання потрібних комбінацій умовних елементів *and* і *or* можна змішувати різні умови і групувати елементи умов виконання правил так, як цього вимагає логіка їх застосування.

Умовний елемент *exists.*

Дозволяє визначати, чи існує хоча б один набір даних (фактів), який відповідає заданій умові. Синтаксис:

<умовний-елемент-exists> ::= (exists <умовний елемент>+)

Умовний елемент *exists* дозволяє виконувати успішне зіставлення з шаблонами з урахуванням наявності щонайменше одного факту, що узгоджується з шаблоном, не беручи до уваги те, яка загальна кількість фактів насправді узгоджується з шаблоном. Це дає можливість створювати єдине часткове узгодження або єдине активування для правила, виходячи з наявності хоча б одного факту з класу фактів.

Умовний елемент *forall.*

Умовний елемент *forall* дозволяє виконувати зіставлення з шаблонами з використанням множини умовних елементів, перевірка яких завершується успішно стосовно кожного входження іншого умовного елемента. Дозволяє визначити, що деяка умова виконується для всіх заданих умовних елементів.

<умовний-елемент-forall> ::= (forall <conditional-element>+)

Для того, щоб перевірка за допомогою умовного елементу *forall* була завершена успішно, кожному факту, що узгоджується з певним шаблоном з переліку <conditional-element>, повинні також відповідати факти зі списку, які узгоджуються з усіма рештою шаблонів, які наявні в умовній частині правила.

Умовний елемент *logical.*

Умовний елемент *logical* дозволяє визначити, що деяка задана умова вико-

нується для всіх заданих умовних елементів і надає механізм підтримки достовірності для створених правилом даних, які відповідають зразкам; також дозволяє вказати, що існування деякого факту залежить від існування іншого факту або групи фактів. Фактично, цей елемент є засобом підтримки істинності, передбаченим в мові CLIPS – умовний елемент *logical* дозволяє створити залежність між фактами, які зіставляються з шаблонами в лівій частині правила, і фактами, введеними в список фактів в результаті виконання його правої частини. Тобто, створюється логічний зв'язок фактів, що активують правило, з фактами, які вводяться у список в результаті його виконання. Тому, якщо відбувається будь-яка зміна фактів, які активують правило, то із списку фактів автоматично виводиться і факт, який був введений в нього в результаті виконання відповідного правила. Таким чином, забезпечується механізм підтримки достовірності для створених правилами фактів.

Його синтаксис:

<умовний-елемент-logical> ::= (logical <conditional-element>+).

Особливості використання умовного елементу *logical* – він не обов'язково повинен включати всі шаблони в лівій частині правила, але обов'язково повинен включати перший умовний елемент в ній, причому кількість умовних елементів *logical* в правилі не може бути більше одного. Крім того, з використанням умовних елементів *not*, він дозволяє зробити факти залежними від відсутності інших фактів. В умовному елементі *logical* можна також перевіряти більш складні умови за допомогою умовних елементів *exists* і *forall* або інших поєднань умовних елементів. Але умовний елемент *logical* діє в усіх відношеннях подібно до умовного елементу *and*, не рахуючи того, що він створює залежності між групами фактів.

Для перевірки залежності між фактами у CLIPS передбачена команда *dependents* з синтаксисом:

(dependents <fact-index-or-address>),

яка дозволяє виявити у списку фактів факти, що логічно пов'язані між собою.

Умовний елемент *test.*

Умовний елемент *test* надає можливості накладення додаткових обмежень на слоти фактів. Синтаксис цього елемента:

<умовний-елемент-test> ::= (test <виклик-функції>)

Елемент *test* задовольняється, якщо викликана в ньому функція повертає значення *TRUE*.

В умовному елементі *test* можна використовувати змінні, вже пов'язані зі своїми значеннями. Усередині елементу *test* можуть бути виконані різні логічні операції – наприклад, порівняння змінних.

Вираз *test* обчислюється кожного разу при наявності нової групи фактів, що задовольняють його умовні елементи. Це означає, що значення умовного елементу *test* буде визначатися кожного разу у випадку, якщо вираз, який його визначає, може бути задоволений більше ніж однією групою даних.

Потрібно мати на увазі, що застосування умовного елементу *test* може стати причиною автоматичного додавання правилом деяких умовних виразів. Крім того, CLIPS може автоматично змінювати порядок умовних елементів *test*.

Порядок виконання роботи:

1. Запустіть CLIPS, відкрийте вбудований редактор і наберіть в ньому код програми (або наберіть її код у будь-якому текстовому редакторі з наступним поміщенням його в середовище CLIPS), в якій передбачити:
   - введення масиву з не менше 20 впорядкованих фактів виду (ocinka Prizvyshche-Imja-Po-Bat'kovi o1 o2 o3 o4), що відображають результати сесії, за допомогою конструктора *deffacts*; при формуванні фактів передбачити наявність у списку студентів, що отримали в результаті складання іспитів сесії незадовільні (2), позитивні (3, 4 і 5), оцінки 4 і 5, а також лише відмінні оцінки;

- використовуючи конструктор *defrule*, записати:
  - роздрук заголовку програми;
  - роздрук списку результатів сесії (тобто введених фактів);
  - правило непереведення, за наявності незадовільної оцінки на сесії, студентів на наступний курс, з додаванням факту про відрахування та виведення на друк повідомлення про відрахування;
  - видалення з подальшого розгляду відрахованих студентів;
  - правило переведення студентів на наступний курс (за відсутності незадовільної оцінки на сесії) з виведенням на друк повідомлення про переведення;
  - правило нарахування стипендії відмінникам та виведення на друк відповідного повідомлення;
  - видалення з подальшого розгляду студентів-відмінників;
  - правило нарахування стипендії переведеним студентам, та виведення на друк відповідного повідомлення;
  - правило ненарахування стипендії переведеним студентам, та виведення на друк відповідного повідомлення;
  - пояснення окремих частин правил, супроводжуючи кожну частину правила відповідним коментарем.

2. Використовуючи встановлення пріоритетності правил, забезпечити потрібну послідовність виконання правил та запис відповідних заголовків блоків виведення.
3. Запишіть набраний код в файл під назвою lab-5-rules.CLP.
4. Введіть набраний код у робочу пам’ять CLIPS; для цього “замаркуйте” (виділіть) написаний код і натисніть набір клавіш CTRL-M (або виберіть пункт Batch Selection з меню Buffer). Для кращого контролю правильності написаного коду можна вводити програму окремими блоками почергово, виділивши відповідний блок програми і ввівши його в пам’ять системи.
5. Якщо додавання правил відбулося з помилкою, CLIPS видрукує у командному

рядку повідомлення з характером помилки; відповідно до вказаної помилки уважно перевірте код у відповідному блоці.

6. Якщо код введено без помилок, у діалоговому вікні з'явиться запрошення у вигляді "CLIPS>".
7. Використовуючи вікна Deffacts Manager та Defrule Manager в меню Browse, переконатися в наявності введених шаблонів фактів та правил у робочій пам'яті системи.
8. Виставте умови трасування роботи програми. Для цього виставте прапорці у вікнах Facts, Rules, Activation випадного меню Watch.
9. Виконайте команду *reset*.
10. Використовуючи команду *facts* (або вікно 1Facts (MAIN) в меню Window), перевірте наявність введення потрібних фактів.
11. Використовуючи команду *rules* (або вікно Defrule Manager в меню Browse), перевірте введення всіх правил.
12. Запустіть виконання програми, використавши команде *run*. Використовуючи трасування її виконання в діалоговому вікні, пересвідчіться у правильності виконання всіх правил програми,.
13. Зніміть прапорці у вікнах Facts, Rules, Activation випадного меню Watch.
14. Роздрукуйте результати виконання програми.
15. Очистіть пам'ять середовища за допомогою команди *clear*.
16. Завантажте записаний код програми з файлу lab-5-rules.CLP і виконайте процедури 5-10.
17. Використовуючи команду Refresh випадного вікна Defrule Manager в меню Browse, відновіть правила програми і запустіть її виконання.
18. Напишіть аналогічну програму для випадку, коли факти, які вводяться конструктором *deffacts*, невпорядковані, та запустіть її на виконання.
19. Напишіть звіт про виконану роботу; при оформленні звіту використовуйте скріншоти результатів окремих етапів роботи.
20. В додатку до звіту представити коди написаних програм.

Контрольні запитання:

1. Які умовні елементи використовуються у лівій частині правил у CLIPS?
2. Що таке зразок (*pattern*) і які особливості він надає лівій частині правила?
3. Що таке обмеження полів і для чого вони найчастіше використовуються при створенні правил?
4. У якій частині правила найчастіше використовуються обмеження полів?
5. Шо являють собою єднальні сполучення і яку роль відіграють?
6. Що таке предикатні обмеження і в яких випадках вони застосовуються?
7. В чому полягає суть символьних обмежень і які особливості на активування правил накладає їх застосування?
8. Яка роль умовних елементів при формуванні умов виконання правил?
9. Які типи умовних елементів використовуються при формуванні антецедентної частини правил?
10. Які можливості надає використання умовного елементу *test*?
11. Як можна перевірити наявність активованих правил у пам'яті системи?
12. Для чого використовується команда *refresh* і як її можна виконати?
13. З якою метою і переважно коли використовуються команди з меню Watch?
14. Як вводяться і для чого використовуються в правилах локальні змінні?
15. В якій частині правила найчастіше вводяться локальні змінні? Як і де вони можуть використовуватися?
16. Яким чином вводяться глобальні змінні в CLIPS?
17. Які два різних типи групових символів використовується в CLIPS і яка між ними різниця?
18. У яких випадках використовуються прості групові символи, а в яких – складені?
19. За допомогою якої функції можна ввести просту або глобальну змінну чи змінити їх значення?
20. Які змінні використовуються у випадку одно-польних чи багато-польних

(складених) змінних?

21. Чи зміняться деякі елементи програми при введенні початкових даних роботи у вигляді впорядкованих і невпорядкованих фактів?

Література:

1. CLIPS. A Tool for Building Expert Systems [Електронний ресурс], 2016. – Режим доступу: https://sourceforge.net/projects/clipsrules/files/CLIPS/6.30/clips_documentation_630.zip/download
2. CLIPS Rule Based Programming Language [Електронний ресурс], 2016. – Режим доступу: https://sourceforge.net/projects/clipsrules/files/CLIPS
3. CLIPS Reference Manual. Volume I. Basic Programming Guide. Version 6.30 March 17th 2015 – 416 p. [Електронний ресурс] – Режим доступу: http://clipsrules.sourceforge.net/documentation/v630/bpg.pdf
4. Джарратано Джозеф. Экспертные системы. Принципы разработки и программирование, 4е изд: Пер. с англ. / Джозеф Джарратано, Гари Райли. – М., Изд. дом “Вильямс”, 2007. – 1152 с.

Додаток 1. Вирази для програмування
(вибір індивідуального завдання – за порядковим номером студента в списку групи)

1. $\sin 1{,}2 - 1/(\cos 1 - 2) - (2e^4 - 4 - |\sin 6^2|)/(2\sin 2{,}4 - 4(\mathrm{tg}\,0{,}4 + \ln 2)/(2{,}5e^{1+\ln 2{,}3}) + \sqrt{5^4 + \sqrt{7^2+1} + \ln 20.5 - 35e^{-\cos 2}}) + (3^3 - e^{7+\sin 3})/(|3e^3 - 2\ln 34| + e^{4-3\sin 2/7\cos 4})$.

2. $(2e^4 - 4 - |\sin 6^2| - (\sin 12 + 1/(\cos 1 - 2)))/(3^3 - e^{7+\sin 3} - 4\ln(2\cos(3+\ln 2) + \sqrt{5^{4\ln 2} + \sqrt{7^{\cos 4}+1} + \ln(\cos 4 + 3)}))) + 4/|5e^{3+\ln 3{,}3} - 2\ln 34| - 2\sin 2*(3^3 - e^{7+\sin 3})$.

3. $\sin(1-\cos 2) - (\ln 2 + 1)/(\cos(\ln 11 - 2)) + (2e^4 - 4 - \cos(12+6^2))/\sqrt{5^{4+\sin 1} + \sqrt{4^3+12} + \ln 1.5} - (\ln 2 + \sin 3 - 2/e^{4-\ln 2} - 3^3 - e^{7+\sin 3})/|3e^3 - 2\ln 34|$.

4. $(2 - 3^3 - e^{7+\sin 3})/(|1 + 3e^3 - 2\ln 34| + e^{4\cos 5 - 3\sin 2/7\cos 4}) + (3^3 - e^{7+\sin 3})/(|3e^3 - 2\ln(3 - \sin 4)| + e^{4-3\sin 2/7\cos 4}) - \sin 1 - 1/(\cos 1 - 2 + 2e^{4\ln 3 + 2\sin 3} - 4 - |\sin 2^{4\mathrm{coc}2 - \ln 3}|)$

5. $\sin 13 - 1/(\cos 1 - 2) + (2e^{4\ln 3 + 2\sin 3} - 4 - |\sin 2^4|) + (3^3 - e^{7+\sin 3})/3 - \sqrt{5^{\cos 4} + \sqrt{7^{2\ln 24}+1} + \ln 10.5}/(|3e^3 - 2\ln 34| + e^{4-3\sin 2/7\cos 4}) + 2e^{2\cos 1}|\ln 4 - 7e^{2\ln 4 - 5\sin 2}|$.

6. $(2 + 3^3 - e^{7+\sin 3})/(|1 + 3e^3 - 2\ln 34| + e^{4\cos 5 + 3\sin 2/7\cos 2}) + \sin 13 + 1/((\cos 1 - 2)/(2e^4 - 4 - |\sin 6^{2(\sin 4 + \ln 23)}|/2\sin 2{,}4 + \sqrt{5^4 + \sqrt{7^2+1} + \ln 20.5 - 35e^{-\cos 2}}))$.

7. $3^3 + e^{7+\sin 3 - 2\ln 4}/\sqrt{5^7 + 2\cos 4 + 1/\ln 20.5} + 11/(|5e^{3+\ln 3{,}3} - 2\ln 34|) - 1/(\sin(1 - 1/(\cos 1 - 2^4) - (2\sin 3 + \cos(2\ln 3 + e^{\mathrm{tg}\,0{,}7}/(2e^{4\ln 3 - 2\sin 3/24\ln 1{,}5} - 4 + |\sin 3^4 - 5|))$.

8 $2e^4 - 4 - |\cos(\sin 6^{2\cos 4})| - (\sin 1 + 23\sin(2 + \ln(3\sin 2) + 1))/((\cos 1 - 2) - 2\ln(-2(3^3 - e^{7+\sin 3})) + \sqrt{5^4 + \sqrt{7^2 + 1} + \ln 20.5} + |5e^{3+\ln 3,3} - 2\ln 34|)$.

9 $(\sin 1 + 3\ln 2/(\cos 1 - 2) + (2e^4 - 4 - |\sin 6^2|))/(2\sin 2{,}4 + \ln(5-\sin 2)) - \sqrt{5^4 + \sqrt{7^{2+\cos 12} + 1} + \ln 22.5 - 35e^{-\cos 2}} + (3^3 - e^{7+\sin 3})/(|3e^3 - 2\ln 34| + e^{4-3\sin 2/7\cos 4})$.

10. $((\sin 1 - \cos 2)/(\cos 1 - 2) + 2e^{2\cos 1}|\ln 4 - 7e^{2\ln 4 - 5\sin 2}|) / (\ln 2 + \sin 3 - 2/e^{4-\ln 2} - 3^3 - e^{7+\sin 3}) - (2e^{4\ln 3 + 2\sin 3} - 4 - |\sin 2^4|)\ (|3e^3 - 2\ln 34|)$.

11. $(2\sin 2{,}4 + \ln(5-\sin 2) + (2e^4 - 4 - |\cos(\sin 6^{2\ln 2})|\ )) / ((2e^4 - 4 - |\cos(\sin 6^{2/\sin 3})|\ ) / ((2\sin 2{,}4 + \ln(5-\sin 2)) - (2e^{4\ln 3 + 2\sin 3} - 4 - |\sin 2^{4--\cos 3}|) / (2 - 3^3 - e^{7+\sin 3})\ )$.

12. $(2 + 3^5 - e^{7+\sin 3})/(|3 + 3e^4 - 2\ln 14| + e^{24\cos 5 + 3\sin 2/7\cos 2}) + (3^3 - e^{7+\sin 3})/(|3e^3 - 2\ln 34| + e^{4-3\sin 2/7\cos 4}) - \sin 10 - 1/(\cos 4 - 2 + 2e^{4\ln 3 + 2\sin 3} - 4 - |\sin 2^{4+\ln 2}|)$.

13. $(\sqrt{4^5 + \sqrt{9^2 + 4} + \ln 17.5 - 4e^{-2\sin 2}} + (3^3 - e^{7+\sin 3})/(|3e^3 - 2\ln 34| + e^{4-3\sin 2/7\cos 4})\ ) + (3/(\sin 0{,}8 - 3/(\ln 2 - 2^4) - \cos(4/(2e^{4\ln 3 - 2\sin 3/24\ln 1{,}5} - 4 + |\sin 3^{4-\mathrm{tg}1} - 5|))))$.

Додаток. 2. Вимоги до виконання та представлення результатів лабораторних робіт.

Протягом семестру обов’язково виконуються п’ять лабораторних робіт із запропонованого в посібнику списку (за вибором викладача).

При виконанні лабораторної роботи студенти повинні

1) уважно вивчити теоретичні відомості, що відносяться до даної лабораторної роботи;
2) незрозумілі питання з’ясувати за допомогою додаткової літератури;
3) продемонструвати роботу написаної програми на комп`ютері згідно з вимогами, вказаними у завданні до лабораторної роботи;
4) оформити звіт.
5) захистити звіт.

При захисті лабораторної роботи студенту необхідно представити звіт з лабораторної роботи, продемонструвати роботу програми та відповісти на контрольні питання.

Звіт складається з:

- титульної сторінки;
- змісту;
- теми роботи;
- мети;
- завдання до роботи;
- коротких теоретичних відомостей, необхідних для виконання роботи;
- тексту написаної програми (у додатку);
- опис отриманих результатів роботи програми з врахування представ-

лених у завданні до неї вимог;

- висновків.

У представленому звіті повинні бути представлені усі потрібні відомості, які потрібні для виконання роботи та її захисту, з використанням необхідного графічного матеріалу (зокрема, скріншотів вузлових моментів виконання роботи), які пояснюють та ілюструють результати роботи написаної програми, у т. ч. при забезпеченні виконання поставлених у завданні вимог.

Оформлення звіту виконувати з дотриманням прийнятих в університеті нормативних документів щодо оформлення звітної документації.

Терміни захисту виконаних робіт показані в табл. 1. У випадку невчасного виконання та захисту лабораторної роботи підсумкові бали за її виконання знижуються на 50%.

Таблиця 1. Терміни захисту виконаних робіт:

Робота №	1	2	3	4	5
Термін захисту, тиждень семестру	3	6	9	12	15

Розподіл балів, що присвоюється студентам для заліку.

Сумарна оцінка контролю успішності протягом семестру виставляється за 100-бальною шкалою згідно критеріїв, встановлених МОН, і включає:

- оцінку виконання завдань на контрольних роботах визначення ступеня засвоєння матеріалу змістових модулів (письмові завдання) (максимум 40 балів):

- оцінку виконання, представлення та захисту звітів лабораторних робіт (максимум 50 балів); при оцінюванні лабораторної роботи враховується підготовка до виконання лабораторної роботи, хід виконання лабораторної роботи, оформлення звіту, отримані результати та захист звіту про виконану лабораторну роботу.
- оцінку активності студента протягом семестру (максимум 10 балів).

Таблиця 2.

Поточне тестування та самостійна робота		**Лабораторні роботи**	**Оцінка активності протягом семестру**	**Сума**
Модуль 1	**Модуль 2**			
Т1 – Т8	**Т9 – Т16**			
20	20	50	10	**100**