

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни

“Функціональне та логічне програмування”

“Функціональне програмування”

для студентів спеціальності

121 «Інженерія програмного забезпечення»

освітня програма

«Програмне забезпечення систем»

усіх форм навчання

Методичні вказівки до лабораторних робіт з дисципліни „Функціональне та логічне програмування” “Функціональне програмування” для студентів спеціальності 121 «Інженерія програмного забезпечення» освітня програма «Програмне забезпечення систем» усіх форм навчання /Укл.:І.В.Левада, Л.Ю.Дейнега.-Запоріжжя: ЗНТУ, 2016.-78с.

Укладачі: І.В. Левада, старш. викладач,
Л.Ю. Дейнега, старш. викладач.

Рецензент: А.В. Пархоменко, доц., к. т. н.

Відповідальний за випуск: старший викладач І.В. Левада

Затверджено на засіданні

НМК КНТ- факультету

від 23 жовтня 2016р.

Затверджено на засіданні

кафедри ПЗ протокол №5

від 17 листопада 2016 р.

ЗМІСТ

ПЕРЕДМОВА.....	7
1 ЛАБОРАТОРНА РОБОТА №1.....	8
Інтерпретатор і пакетна організація середовища Common Lisp ..	8
1.1 Мета роботи	8
1.2 Завдання на лабораторну роботу	8
1.3 Опис середовища функціонування PRACTICAL COMMON LISP	9
1.4 Методичні вказівки до лабораторної роботи	9
1.5 Зміст звіту	9
1.6 Контрольні питання	9
2 ЛАБОРАТОРНА РОБОТА №2.....	10
Робота у оболонці SLIME	10
2.1 Мета роботи	10
2.2 Завдання на лабораторну роботу	10
2.3 Опис середовища функціонування PRACTICAL COMMON LISP	11
2.4 Методичні вказівки до лабораторної роботи	11
2.5 Зміст звіту	11
2.6 Контрольні питання	12
3 ЛАБОРАТОРНА РОБОТА №3.....	13
Рекурсія на списках	133
3.1 Мета роботи	133
3.2 Завдання на лабораторну роботу	133
3.3 Опис середовища функціонування PRACTICAL COMMON LISP	13
3.4 Методичні вказівки до лабораторної роботи	13
3.5 Зміст звіту	144
3.6 Контрольні питання	14
ЗАВДАННЯ ДО ЛАБОРАТОРНИХ РОБІТ №3, 4.....	155
Тема: Обробка списків	15
4 ЛАБОРАТОРНА РОБОТА №4.....	18
Обробка списків спеціальними формами циклів.....	18
4.1 Мета роботи	18
4.2 Завдання на лабораторну роботу	18
4.3 Опис середовища функціонування PRACTICAL COMMON LISP	18
4.4 Методичні вказівки до лабораторної роботи	18
4.5 Зміст звіту	19
4.6 Контрольні питання:	19
5 ЛАБОРАТОРНА РОБОТА №5.....	20

Рядки та знаки	20
5.1 Мета роботи	20
5.2 Завдання на лабораторну роботу	20
5.3 Опис середовища функціонування PRACTICAL COMMON LISP	20
5.4 Методичні вказівки до лабораторної роботи	20
5.5 Зміст звіту	20
5.6 Контрольні питання	21
ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №5	22
ТЕМА: Обробка рядків	22
6 ЛАБОРАТОРНА РОБОТА №6.....	24
Робота з файлами	24
6.1 Мета роботи	24
6.2 Завдання на лабораторну роботу	24
6.3 Опис середовища функціонування PRACTICAL COMMON LISP.....	255
6.4 Методичні вказівки до лабораторної роботи	25
6.5 Зміст звіту	25
6.6 Контрольні питання	25
ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №6	266
ТЕМА: Робота з файлами.....	26
Література.....	29
Основна.....	29
Додаткова.....	29
ДОДАТОК А	29
Інтерпретатор і пакетна організація середовища Common Lisp	30
А.1 Склад дистрибутиву Practical Common Lisp	30
А.2 Запуск Practical Common Lisp та вихід з нього	30
А.3 Робота з інтерпретатором	30
А.3.1 Вираз – виклик функції.....	31
А.3.2 Вираз – складне дане список	31
А.3.3 Вираз - змінна	31
А.3.4 Функція QUOTE	32
А.3.5 Вираз – означення функції.....	33
А.4 Середовище Common Lisp	33
А.4.1 Поняття середовища.....	33
А.4.2 Поняття пакету.....	33
А.4.3 Створення пакету та його символів	343
А.4.4 Робота з символами пакетів.....	35
А.4.5 Стандартні пакети	37
ДОДАТОК Б	38

Робота у оболонці Slime	38
Б.1 Система програмування «Lisp in a Box»	38
Б.2 Організація X - вікон та буферів	38
Б.2.1 Поняття буфера та X – вікна	38
Б.2.2 Створення та закриття буферів	39
Б.2.3 Створення X - вікон	40
Б.2.4 Довідкова інформація про буфери	40
Б.3 Опції текстового редактору	43
Б.3.1 Меню текстового редактору	43
Б.4 Команди текстового редактору з клавіатури	47
Б.5 Робота системою програмування	48
ДОДАТОК В	50
Базові функції	50
В.1 Базові функції відбору	50
В.2 Базові функції конструктори	51
В.3 Функції компаратору	51
В.4 Функції предикати	53
В.5 Керуючі спеціальні форми роботи із контекстом	55
В.6 Керуючі спеціальні форми послідовного виконання	56
В.7 Керуючі спеціальні форми розгалуження обчислень	56
ДОДАТОК Г	588
Рекурсія	58
Г.1 Поняття рекурсії	58
Г.1.1 Низхідний метод рекурсії	58
Г.1.2 Висхідна рекурсія	59
ДОДАТОК Д	60
Рекурсивна обробка списків	60
ДОДАТОК Е	61
Спеціальні форми організації циклів	61
Е.1 Спеціальна форма DO	61
Е.2 Спеціальна форма LOOP	63
ДОДАТОК Є	64
Робота з рядками та знаками	64
Є.1 Наслідування функцій	64
Є.2 Типи знаку та рядку	65
Є.3 Функції перетворення типів даних	66
Є.4 Дії над рядками	67
Є.4.1 Функції порівняння рядків та символів	67
Є.4.2 Функції відбору	68

Є.4.3 Заміна знаку рядка	68
Є.4.4 Функція створення рядку	69
Є.4.5 Функції зміни регістру	69
Є.4.6 Функція усікання рядку.....	69
Є.4.7 Функція пошуку.....	69
Є.5 Робота з властивостями символів.....	70
Додаток Ж.....	71
Функції вводу-виводу.....	71
Ж.1 Функції вводу.....	71
Ж.2 Функції виводу.....	71
Додаток З.....	74
Робота з файлами.....	74
3.1 Функції роботи з файлами.....	74
3.2 Приклади роботи з файлами.....	75
3.2.1 Читання даних з файлу.....	75
3.2.2 Запис даних у файлу.....	76

ПЕРЕДМОВА

У дисципліні «Функціональне програмування» вивчається мова „Practical Common Lisp”, яка є однією з програмних реалізацій стандарту COMMON LISP.

Стандарт COMMON LISP - це тільки опис мови, а не канонічна реалізація мови. Для COMMON LISP не існує канонічної програмної реалізації, як у інших сучасних мовах програмування. Програми користувачів, що застосовують тільки можливості дозволені стандартом, можуть працювати на будь-якій реалізації COMMON LISP.

Первісно мова COMMON LISP існувала тільки, як діалект Lisp – сімейства. Для тих, хто знає можливості хоча б однієї мови з Lisp – сімейства, відомі переваги таких мов перед мовами традиційного програмування. Lisp вбирає в себе все найкраще, що має програмування на сьогодні. Ніякі вдосконалення мов C++, Perl, Python не дозволять додати в них можливості для реалізації всіх методів програмування, властиві мовам Lisp-сімейства.

COMMON LISP базується на функціональній парадигмі програмування. Проте, мова не відноситься до „чистих” мов, що суворо придержуються принципів функціонального програмування. Сучасне практичне програмування часто вимагає реалізації методів традиційного програмування.

Метою вивчення дисципліни є одержання знань і практичних навиків роботи з сучасною мовою, що базується на функціональній парадигмі.

Мову COMMON LISP призначено для використання у практичній діяльності кваліфікованими програмістами. Її використовують для розв’язку інженерних задач, задач штучного інтелекту; на ній реалізують WEB – застосування, текстові редактори, графічні інтерфейси користувача, спеціалізовані мови вищого рівня, програми спілкування з реляційними базами даних, тощо. У мережі Internet існує багато бібліотек з відкритим для користувачів кодом. Вміст бібліотек дозволяє значно розширяти можливості Common Lisp. Мобільність мови дозволяє легко розширювати її.

1 ЛАБОРАТОРНА РОБОТА №1

Інтерпретатор і пакетна організація середовища COMMON LISP

1.1 Мета роботи

Мета роботи:

- отримати уяву про роботу інтерпретатора;
- навчитися призначати змінній значення та одержувати значення змінних, утворювати та виконувати функції;
- ознайомитись з пакетною організацією середовища.

1.2 Завдання на лабораторну роботу

1.2.1 Вивчити матеріал з додатку А “Інтерпретатор і пакетна організація середовища Common Lisp”.

1.2.2 Дослідити роботу інтерпретатору, якщо на вхід інтерпретатору поступає виклик стандартної функції. Застосуйте для цього різні функції відомі вам з лекцій.

1.2.3 Дослідити роботу інтерпретатору, якщо на вхід інтерпретатору поступають різні типи даних, змінні. Застосуйте для цього приклади з додатку А.

1.2.4 Дослідити роботу інтерпретатору, якщо на вхід поступає означення функції користувача. Застосуйте для цього приклад з додатку А. Утворіть своє означення функції.

1.2.5 Утворити у середовищі пакет з ім'ям Proba. Інтернувати в пакет декілька символів. Привласнити символам значення.

1.2.6 Утворити у середовищі пакет з ім'ям Work. Інтернувати в пакет декілька символів, серед яких є один символ з Print – ім'ям символу пакету Proba. Привласнити символам значення. Перевірити значення всіх символів з пакетів Proba та Work. Які значення мають символи, що мають різні та однакові імена?

1.2.7 Зробити пакет Work поточним. Зробити пакет Proba поточним. Як перевірити, що певний пакет поточний.

1.2.8 Зробити символ пакету Proba зовнішнім. Як взяти, які пакети використовують зовнішні символи поточного пакету?

1.2.9 Перевірити чи входить символ у певний пакет?

1.2.10 Перевірити чи існує пакет з певним ім'ям.

1.2.11 Взяти, яким пакетам належить певний символ.

1.3 Опис середовища функціонування PRACTICAL COMMON LISP

Для роботи системи програмування PRACTICAL COMMON LISP треба мати комп'ютер сумісний з комп'ютером фірми IBM та одну з операційних систем WINDOWS, UNIX, LINUX.

1.4 Методичні вказівки до лабораторної роботи

Для виконання пунктів 1.2.2 – 1.2.11 треба вивчити матеріал з додатку А та виконати приклади з додатку.

1.5 Зміст звіту

Звіт про виконану роботу на комп'ютері.

1.6 Контрольні питання

1.6.1 Дайте означення середовища COMMON LISP ?

1.6.2 Які дії виконує інтерпретатор?

1.6.3 Поясніть, яке у символу зовнішнє і внутрішнє представлення?

1.6.4 З чого складається ядро пакета?

1.6.5 Які типи символів зберігаються у пакеті?

1.6.6 Яки ви знаєте функції для роботи з пакетами та яке їх призначення?

1.6.7 Які стандартні пакети має COMMON LISP?

1.6.8 Як перетворити рядок у символ пакету?

ЛАБОРАТОРНА РОБОТА №2

Робота у оболонці SLIME

2.1 Мета роботи

Мета роботи:

- вивчити опції оболонки Slime для роботи з Common Lisp;
- вивчити команди текстового редактору Emacs необхідні для роботи з Common Lisp;
- навчитися налагоджувати та виконувати програму на COMMON LISP у оболонці Slime.

2.2 Завдання на лабораторну роботу

2.2.1 Вивчити матеріал з додатку Б “Робота у оболонці Slime”.

2.2.2 Утворити декілька вікон на екрані різними опціями меню; командами редактору з клавіатури.

2.2.3 Утворити буфери різними командами.

2.2.4 Змінити зв’язок між поточним буфером і вікном різними способами.

2.2.5 Завантажити файл у буфер різними способами, використовуючи роботу з міні буфером.

2.2.6 Підготувати у блоктоті файл з програмою, яка рахує суму елементів числового списку.

Означення функції:

```
(Defun Sum (L S) (cond ((Null L) S) (T (Sum (cdr L) (+ s (car L))))))
```

Виклик функції.

```
(sum '(1 3 6) 0)
```

Завантажити програму на вхід інтерпретатора та виконати її.

2.2.7 Зберегти програму з пункту 2.2.6 у файл, компілювати її, завантажити на вхід інтерпретатора та виконати.

2.2.8 Змінити програму з пункту 2.2.6 так, щоб вона рахувала кількість елементів списку. Завантажити програму на вхід інтерпретатора та виконати її.

2.3 Опис середовища функціонування PRACTICAL COMMON LISP

Для роботи системи програмування PRACTICAL COMMON LISP треба мати комп'ютер сумісний з комп'ютером фірми IBM та одну з операційних систем WINDOWS, UNIX, LINUX.

2.4 Методичні вказівки до лабораторної роботи

Для виконання пунктів 2.6 та 2.8 застосовуються стандартні функції: Car, Cdr, Null, Cond.

Функція Car одержує із списку перший елемент не змінюючи сам список.

Функція Cdr одержує хвіст списку – список без першого елемента. Сам список не змінюється.

Функція Null перевіряє чи порожній список, якщо порожній то повертає True, інакше повертає Nil.

Рекурсивна функція Sum перевіряє, чи порожній вхідний список L. Якщо список не порожній, то функція викликає себе знов з хвостом списку (Cdr L) та з сумою $S=0 + (Car L)$ перший елемент списку.

Дії повторюються до тих пір поки список не стане порожнім, а у змінній і не накопичиться сума елементів.

Якщо список стане порожнім то повертається сума всіх елементів списку.

Нижче показано як працює функція Sum:

Виклик функції; Список;	Сума
Sum; L=(1 3 6);	S=0
Sum; L=(3 6);	S=1
Sum; L=(6);	S=4
Sum; L=();	S=10

2.5 Зміст звіту

5.1 Усний звіт про роботу на комп'ютері з відповіддю на питання за матеріалом додатку Б.

2.6 Контрольні питання

2.6.1 Чому X - вікно називають фреймом?

2.6.2 Чи може один буфер мати декілька вікон?

2.6.3 Що таке точка у буфері?

2.6.4 Які команди застосовують при роботі з міні буфером?

2.6.5 Як налагодити зв'язок з оболонкою Slime, якщо зв'язок розірвано?

2.6.6 Для чого застосовують команду OPEN DIRECTORY?

2.6.7 Як включити структурний редактор?

2.6.8 Що робить команда CLOSE?

2.6.9 Як очистити буфер? Що очищується при цьому середовище Common Lisp?

2.6.10 Як одержати довідку про певну функцію Common Lisp?

3 ЛАБОРАТОРНА РОБОТА №3

Рекурсія на списках

3.1 Мета роботи

Метою роботи є:

- отримати знання про базові функції COMMON LISP;
- навчитись писати програми, що оброблюють списки рекурсивним методом;
- навчитись обирати метод рекурсії відповідно завданню.

3.2 Завдання на лабораторну роботу

3.2.1 Вивчити матеріал з додатку В “Базові функції”.

3.2.2 Вивчити матеріал з додатку Д „Рекурсія”.

3.2.3 Вивчити матеріал з додатку Е „Рекурсивна обробка списків”.

3.2.4 Обрати варіант завдання за номером у журналі. Обрати завдання на сторінці 14. Тема : „Обробка списків”.

3.2.5 Обрати метод рекурсії для написання програми.

3.2.6 Написати та налагодити програму за обраним рекурсивним методом. Виконати програму і отримати результати.

3.3 Опис середовища функціонування PRACTICAL COMMON LISP

Для роботи системи програмування PRACTICAL COMMON LISP треба мати комп'ютер сумісний з комп'ютером фірми IBM та одну з операційних систем WINDOWS, UNIX, LINUX.

3.4 Методичні вказівки до лабораторної роботи

3.4.1 Аналізуйте завдання і виявіть, який метод рекурсії(висхідний чи низхідний) зручніше застосувати до написання програми, що обробляє списки.

3.4.2 Використовуйте при програмуванні базові функції, які ви вивчили.

3.5 ЗМІСТ ЗВІТУ

Звіт про роботу повинен містити:

3.5.1 Мету роботи.

3.5.2 Завдання.

3.5.3 Програму.

3.5.4 Вхідні дані та результати роботи програми.

3.5.5 Базові функції.

3.6 Контрольні питання

3.6.1 Яке зовнішнє подавання списку та яке подавання списку у середовищі?

3.6.2 Чому функція LIST має важливе значення? Пояснити.

3.6.3 До яких типів об'єктів застосовують функції CAR та CDR?

3.6.4 Що робить функція CONS і до яких типів об'єктів можна застосовувати її?

3.6.5 Що таке функціональний та імперативний методи програмування?

3.6.6 Який метод програмування ви застосували у програмі?

3.6.7 Які ви знаєте функції предикати та яка їх дія? Для чого використовують у програмі функції предикати?

3.6.8 Що таке спеціальна форма? Які типи спеціальних форм ви знаєте?

3.6.9 Назвіть керуючі спеціальні форми та їх дію.

3.6.10 Поясніть, що таке рекурсивна функція та рекурсивні типи даних.

3.6.11 Які типи рекурсивних методів написання функції ви знаєте?

3.6.12 Наведіть приклад завдання, для якого краще застосовувати висхідний метод рекурсії.

ЗАВДАННЯ ДО ЛАБОРАТОРНИХ РОБІТ №3, 4

Тема: Обробка списків

1 Написати і налагодити програму, котра вставляє у числовий список після парних чисел 1, а після непарних чисел суму елементів вхідного списку.

2 Написати і налагодити програму, котра замінює у числовому списку всі 2 на 5, а всі мінімальні елементи списку на 10.

3 Нехай є список студентів у групі. Відомий також список студентів групи, які склали сесію на відмінно і з однією четвіркою. Визначити % (відсоток) відмінників у групі, а також % (відсоток) , студентів, що мають одну четвірку.

4 Написати і налагодити програму, яка перевіряє, чи однакові два S-вирази. Однакові S - вирази мають однакову структуру і на однакових місцях мають однакові атоми. Якій стандартній функції відповідає ваша програма?

5 Дана послідовність чисел $a_1 \dots a_n$. Якщо в результаті заміни від'ємних членів послідовності їх квадратами, члени будуть утворювати послідовність яка зменшується, то одержати суму членів вихідної послідовності, в протилежному випадку одержати їх добуток.

6 Дан список цілих чисел $a_1 \dots a_n$. Одержати список, всі члени якого отримані таким засобом: члени старого списку з парними номерами помножити на $\max(a_1 \dots a_n)$, а члени з непарними номерами залишити без зміни.

7 Дан список чисел $a_1 \dots a_n$. Одержати список чисел $b_1 \dots b_n$, де b_i – середнє арифметичне всіх членів послідовності $a_1 \dots a_n$, крім a_i .

8 Дано два списку дійсних чисел $(x_1 \dots x_n)$ і $(y_1 \dots y_n)$. Одержати список елементами якого будуть $Z_i = \max(x_1 \dots x_n) - x_i * y_i$.

9 Дан список дійсних чисел $(a_i, a_2, \dots, a_{n-1}, a_n)$. Одержати: кількість двох сусідніх додатних чисел; кількість сусідніх чисел різних знаків.

10 Дан список чисел $(r_1 \dots r_n)$. Одержати суму $S = x_1 * y_1 + \dots x_n * y_n$, де $x_1 \dots x_p$ – від'ємні члени послідовності $(r_1 \dots r_n)$, узяті в порядку їх слідування, а $y_1 \dots y_n$ – невід'ємні члени, узяті в зворотному порядку. Кількість від'ємних і невід'ємних чисел у списку однакова.

11 Дан список дійсних чисел. Поміняти місцями в цьому списку: найбільший і найменший елемент; найбільший і останній елемент.

12 Дано 2 списки цілих чисел $(a_1 \dots a_n)$ і $(b_1 \dots b_n)$. Перетворити список $(b_1 \dots b_n)$ за правилом: якщо $a_i \leq 0$, то b_i збільшити в 10 разів; інакше b_i замінити нулем.

13 Дан список дійсних чисел $(a_1 \dots a_n)$. Потрібно помножити всі елементи списку на квадрат його найменшого члена, якщо $a_1 \geq 0$, і на квадрат його найбільшого члена, якщо $a_1 < 0$.

14 Дан список дійсних чисел $(a_1 \dots a_n)$. Запишіть список в зворотному порядку, якщо він впорядкований по спаданню, або по зростанню. В протилежному випадку вилучити зі списку ті елементи, порядкові номери яких кратні 4. Зберегти попередній порядок залишених членів.

15 Вилучити зі списку всі підсписки з непарними порядковими номерами, а елементи всіх підписків з парними порядковими номерами записати в зворотному порядку.

16 Дан узагальнений список виду: $((a_1 \dots a_n) \dots (b_1 \dots b_n))$. Сформувати список елементами якого є підсписки з парних елементів відповідних початкових підписків.

17 Дан список натуральних чисел $(a_1 \dots a_n)$. Вибрати перший підписок $(a_{i_1} \dots a_{i_k})$ з елементів списку, які йдуть підряд так, щоб $a_{i_1} + \dots + a_{i_k} > m$, де m вводиться з клавіатури. Якщо такий підписок вибрати неможливо, то потрібно сповістити про це.

18 В даному списку дійсних чисел $(a_1 \dots a_n)$ вибрати підписок з елементів списку, які йдуть підряд, найбільшої довжини і щоб його елементи зростали.

19 Дан список цілих чисел $(a_1 \dots a_n)$. Визначити чи є цей список періодичним, тобто чи можливо його одержати повторенням деякої своєї початкової частини елементів.

20 Відома гра на вгадування слів, що складається із тих самих літер, що і слово-зразок. Нехай даний список слів, де кожне слово подано у вигляді підписку літер. Приймавши, що перший підписок – це слово-зразок, вибрати ті підписки (слова), які можуть бути одержані зі зразка.

21 Дан узагальнений список виду: $((a_1 \dots a_n) \dots (b_1 \dots b_n))$, в одному підписку якого є рівно два однакових елементи. Знайти індекси цих елементів в підписку, а також індекс цього підписку в узагальненому списку.

22 Щоб зашифрувати текст можна замінити його кожну літеру наступною за нею по алфавіту. Літера "Я" замінюється на "А". Нехай текст і алфавіт задані у вигляді списків. Зашифруйте текст і результат видайте на екран.

23 Розшифруйте текст, зашифрований за способом, вказаним у завданні 22. Результат виведіть на екран.

24 Дан узагальнений список виду: $((a_1 \dots a_n) \dots (b_1 \dots b_n))$. Сформувати список, елементи якого підписки узагальненого списку, записані в зворотному порядку, а останній елемент кожного підписку сума елементів підписку.

4 ЛАБОРАТОРНА РОБОТА №4

Обробка списків спеціальними формами циклів

4.1 Мета роботи

Мета роботи:

- отримати знання про спеціальні форми COMMON LISP, що організують цикли;
- навчитись писати програми, які оброблюють списки за допомогою циклів;
- навчитись обирати спеціальні форми відповідно завданню.

4.2 Завдання на лабораторну роботу

4.2.1 Вивчити матеріал з додатку Ж “Спеціальні форми організації циклів”.

4.2.2 Обрати завдання з лабораторної роботи № 3. на сторінці 14.

4.2.3 Написати та налагодити за цим завданням дві програми, що реалізують завдання за допомогою спеціальної форми DO та спеціальної форми LOOP. Виконати програми і отримати результати.

4.3 Опис середовища функціонування PRACTICAL COMMON LISP

Для роботи системи програмування PRACTICAL COMMON LISP треба мати комп'ютер сумісний з комп'ютером фірми IBM та одну з операційних систем WINDOWS, UNIX, LINUX.

4.4 Методичні вказівки до лабораторної роботи

4.4.1 Завдання до лабораторної роботи розташовані на сторінці 11. Тема „Обробка списків”.

4.4.2 Використовуйте при програмуванні базові функції, які ви вивчали.

4.4.3 Аналізуйте реалізовані вами програми і виявіть, яку спеціальну форму зручніше застосувати до написання програми, що ви виконали.

4.5 Зміст звіту

Звіт про роботу повинен містити:

4.5.1 Мету роботи.

4.5.2 Завдання.

4.5.3 Програми.

4.5.4 Вхідні дані та результати роботи програми.

4.5.5 Спеціальні форми організації циклів.

4.6 Контрольні питання

4.6.1 Яка форма запису спеціальної форми DO?

4.6.2 Як вийти з DO циклу ?

4.6.3 Скільки діють зв'язки між локальними змінними та значеннями, що вказані у циклу DO?

4.6.4 Для чого вживається форма LOOP?

4.6.5 Як вийти з форми LOOP?

4.6.6 Для чого застосовується функція RETURN?

4.6.7 Чи можна вийти з циклу LOOP за формою COND?

4.6.8 Чим відрізняється вихід з рекурсії від виходу з циклу DO?

5 ЛАБОРАТОРНА РОБОТА №5

Рядки та знаки

5.1 Мета роботи

Мета роботи:

- отримати знання про функції роботи з рядками та знаками;
- навчитись писати програми, які оброблюють рядки.

5.2 Завдання на лабораторну роботу

5.2.1 Вивчити матеріал з додатку И “Робота з рядками та знаками”.

5.2.2 Обрати за своїм варіантом завдання у розділі „Завдання до лабораторної роботи №5 на сторінці 20. Тема: „Обробка рядків”.

5.2.3 Написати та налагодити програму, що реалізує завдання.

5.2.4 Виконати програму і отримати результати.

5.3 Опис середовища функціонування PRACTICAL COMMON LISP

Для роботи системи програмування PRACTICAL COMMON LISP треба мати комп'ютер сумісний з комп'ютером фірми IBM та одну з операційних систем WINDOWS, UNIX, LINUX.

5.4 Методичні вказівки до лабораторної роботи

4.1 Використовуйте при програмуванні функції, які ви вивчали.

4.2 Аналізуйте завдання і виявіть, який метод створення програми зручніше застосувати до вашого завдання.

5.5 ЗМІСТ ЗВІТУ

Звіт про роботу повинен містити:

5.5.1 Мету роботи.

5.5.2 Завдання.

5.5.3 Програму.

5.5.4 Вхідні дані та результати роботи програми.

5.6 КОНТРОЛЬНІ ПИТАННЯ

5.6.1 Які ви знаєте батьківські типи даних для типу даних string?

5.6.3 Як подається тип даних знак?

5.6.3 Які групи функцій для роботи з рядками та знаками ви вивчили?

5.6.4 Для яких типів даних можна застосовувати функцію COERCE?

5.6.5 Як можна перетворити рядок у символ і символ у рядок?

5.6.6 Як можна перетворити рядок у знак і знак у рядок?

5.6.7 Чим можна порівнювати рядки; символи?

5.6.8 Як змінити рядок?

5.6.9 Як створити рядок з однакових елементів?

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №5

ТЕМА: Обробка рядків

1 Написати і налагодити програму, яка перевіряє, чи є вираз, що вводиться з клавіатури у вигляді рядку, S виразом.

2 Написати і налагодити програму, що замінює слово у рядку на інше слово. Запрограмувати можливість продовжувати заміну у рядку. Заміну і продовження пошуку виконувати за запитом користувача.

3 Написати і налагодити програму, яка формує за шаблоном: „іменник дієслово прийменник іменник” речення з слів словника. Словник зберігається у середовищі мови Лісп.

4 Написати і налагодити програму, яка з речень, що за записані шаблоном : прикметник іменник дієслово іменник, формує словник у середовищі Лісп.

5 Написати і налагодити програму , яка на основі шаблона : діяч, дія, об'єкт на який направлена дія, формує речення . Словник з семантичними відмінками задавати у середовищі.

6 Написати і налагодити програму, що вставляє у текст слово після вказаного слова. Продовження пошуку слова для вставки слова виконувати по запиту користувача.

7 Написати і налагодити програму , яка стискає текст, що вводиться з консолі за наступним принципом: декілька, що ідуть підряд однакових символів замінити ознакою стиснення, кількістю стиснутих символів і символом. Стиснення виконувати лише у тому разі, якщо воно дає економію.

8 Написати і налагодити програму , яка розгортає стиснутий текст за принципом, вказаному у завданні 7. Результати вивести на екран.

9 Написати і налагодити програму , яка розпізнає в тексті, що вводиться з консолі слова, які мають одне й те ж сполучення з букв. Результати вивести на екран.

10 Написати і налагодити програму , яка знаходить в тексті, що вводиться з консолі, числівники в називному відмінку. Результат вивести на екран. Словник розмістити у середовищі Лісп.

11 Написати і налагодити програму , яка підраховує частоту використання голосних в тексті, що вводиться з консолі, де частота це відношення кількості голосних до кількості літер в тексті. Результати вивести на екран. Оформити вивід заголовком.

12 Написати і налагодити програму , яка перетворює форму запису речення вказаного типу в предикатну форму, що використовується у мові ПРОЛОГ. Тип речень: “іменник, дієслово, прийменник, іменник”.

13 Написати і налагодити програму , яка перетворює форму запису речення вказаного типу в предикатну форму, що використовується у мові ПРОЛОГ. Тип речень: прикметник, іменник, дієслово, іменник.

14 Написати і налагодити програму , яка підраховує частоту використання шиплячих приголосних в тексті. Текст вводиться з консолі. Частота – це відношення кількості шиплячих до кількості літер в тексті. Результати вивести на екран. Оформити вивід заголовком.

15 Написати і налагодити програму , яка аналізує синтаксис фактів записаних у формі факту на Пролозі. Факти вводити з клавіатури рядком. Опис предикату для факту задавати рядком.

16 Написати і налагодити програму, яка формує рядок у вигляді опису предикату в секції PREDICATES на Мові ПРОЛОГ за введеним фактом.

17 Написати і налагодити програму, яка аналізує синтаксис англійських речень. Аналіз виконувати за шаблоном: артикль, іменник, дієслово, прийменник, іменник.

18 Написати і налагодити програму, яка будуватиме синтаксично вірні англійські речення типу: The book is on the table. Словник (слова з частинами мови) задавати в середовищі Лісп.

19 Написати і налагодити програму, яка підбиратиме прикметники до іменників тексту, що вводяться з консолі за родом, числом та відмінком. Словник прикметників разом з характеристиками задавати в середовищі Лісп.

20 Написати і налагодити програму, яка аналізуватиме синтаксис означення функції на мові LISP, яке введено у вигляді рядка. У описі повинні бути всі компоненти означення функції.

21 Написати і налагодити програму, яка знаходить в тексті, що вводиться з консолі, дієслова в минулому часі. Результат вивести на екран в два стовпчики. Словник розмістити у середовищі Лісп.

22 Написати і налагодити програму, яка підбиратиме прийменники до іменників тексту, що вводяться з консолі за відмінком. Словник прикметників разом з характеристиками задавати в середовищі Лісп.

23 Написати і налагодити програму, яка знаходить в тексті, що вводиться з консолі, іменники чоловічого роду, у множині та називного відмінку. Результат вивести на екран в два стовпчики. Словник розмістити у середовищі Лісп.

ЛАБОРАТОРНА РОБОТА №6

Робота з файлами

6.1 Мета роботи

Мета роботи:

- отримати знання про функції вводу - виводу;
- отримати знання про функції роботи з файлами;
- навчитись писати програми, що оброблюють рядки.

6.2 Завдання на лабораторну роботу

6.2.1 Вивчити матеріал з додатку К “Функції вводу-виводу”.

6.2.2 Вивчити матеріал з додатку Л “Робота з файлами”.

6.2.3 Обрати за своїм варіантом завдання у розділі „Завдання до лабораторної роботи №6 на сторінці 24. Тема: „Робота з файлами”.

- 6.2.4 Написати та налагодити програму, що реалізує завдання.
- 6.2.5 Виконати програму і отримати результати.

6.3 Опис середовища функціонування PRACTICAL COMMON LISP

Для роботи системи програмування PRACTICAL COMMON LISP треба мати комп'ютер сумісний з комп'ютером фірми IBM та одну з операційних систем WINDOWS, UNIX, LINUX.

6.3 Методичні вказівки до лабораторної роботи

- 6.4.1 Використовуйте при програмуванні функції, які ви вивчали.
- 6.4.2 Аналізуйте завдання і виявіть, який метод створення програми зручніше застосувати до вашого завдання.

6.5 Зміст звіту

Звіт про роботу повинен містити:

- 6.5.1 Мету роботи.
- 6.5.2 Завдання.
- 6.5.3 Програму.
- 6.5.4 Вхідні дані та результати роботи програми.

6.6 Контрольні питання

- 6.6.1 Які ви знаєте функції вводу? Охарактеризуйте їх.
- 6.6.2 Які ви знаєте функції виводу? Дайте їм характеристику.
- 6.6.3 Як знайти довжину файлу?
- 6.6.4 Яка функція дозволяє встановити покажчик на певний байт?
- 6.6.5 Які ви знаєте методи відкриття файлу?
- 6.6.6 Які ключові параметри існують для відкриття файлу при читанні файлу?
- 6.6.7 Які параметри існують для відкриття файлу при запису у файл?
- 6.6.8 Як працювати з бінарними файлами?

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №6

ТЕМА: Робота з файлами

1 Написати програму, що організує в файлі довідник тлумачення слів. Довідник відсортований за алфавітом. Довідник повинен мати таку організацію, що дозволяє отримати прямим методом доступу групу слів на кожну літеру алфавіту.

2 Написати програму, що отримує по заданому слову його тлумачення з файлу “Тлумачний словник”. Словник організований як в задачі 1. Він використовує прямий метод доступу до групи слів на першу букву слова.

3 Написати програму, яка корегує файл “Тлумачний словник”, організований як в задачі 1. Виконати такий вид корегування елементів тлумачного словника: корегування слова або тлумачення.

4 Написати програму, що корегує файл “Тлумачний словник”, організований як в задачі 1. Виконати такий вид корегування елементів тлумачного словника: вилучання слова і його тлумачення.

5 Написати програму, що корегує файл “Тлумачний словник”, організований як в задачі 1. Виконати такий вид корегування елементів тлумачного словника: вставку в словник нового слова і його тлумачення.

6 Написати програму, що реалізує функцію доповнення файлу. Опитати ім'я файлу. Якщо файл існує, то прочитати вміст файлу, вивести на екран вміст файлу і дописати файл уведеними даними.

7 Написати програму, що читає вміст файлу, вводить слово, яке треба замінити і слово для заміни. Знаходить у тексті та замінює всі вказані слова. Вміст файлу зберегти на диску.

8 Написати програму, що вставляє у текст файлу певне слово після вказаного. Повторити вставку для всіх знайдених слів у тексті. Вміст файлу зберегти на диску.

9 Написати програму, що вводить слово і вилучає його у тексті файлу. Дії повторити стільки раз, скільки слово зустрічається. Вміст файлу зберегти на диску.

10 Написати програму, що порівнює вміст двох файлів. Програма порівнює файли за байтами. Якщо файли мають різну довжину видати повідомлення. У випадку збігу кількості байтів видається повідомлення з вказаними іменами файлів, номерами байтів у файлі, їх вмістом у бінарному виді.

11 Написати програму, що вводить дані про студентів, які здали сесію на певному факультеті. Кожен запис має такий зміст : N групи, Прізвище та ініціали, N залікової книжки, Предмет, Оцінка, ..., Предмет, Оцінка. Записи повинні бути фіксованої довжини за рахунок того, що кожне поле фіксованої довжини.

12 Написати програму, яка підготує загальні відомості за групами факультету для файлу описаному в завданні 11. Загальні відомості повинні мати такі дані: група, відмінно - кількість студентів; добре - кількість студентів; задовільно - кількість студентів; незадовільно - кількість студентів.

13 Написати програму, яка сортує дані з файлу про студентів, вказані у завданні 11, по прізвищу (в середині групи) і записує їх в файл в отриманому порядку. Групи повинні також розміщуватись в порядку зростання номера групи.

14 Написати програму , яка корегує дані з файлу про студентів, описані в завданні 11 або доповнює відомостями про нового студента.

15 Написати програму , що вводить дані в файл реєстратури: прізвище та адресу. Ввід даних для кожного пацієнта та їх запис у файл здійснювати за реєстраційним номером. Проконтролювати реєстраційний номер, котрий не повинен виходити за можливі межі номерів. Передавати для нового пацієнта перший вільний номер у файлі.

16 Написати програму, що вводить дані в файл діагнозів. Для кожного пацієнта зберігати прізвище, діагнози трьох спеціалістів. Ввід даних для кожного пацієнта та їх запис у файл здійснювати за реєстраційним номером.

17 Написати програму, що рахує у файлі реєстратури за прізвищем, адресою і їх комбінацією кількість пацієнтів, а результати видає на екран. Вміст файлу у завданні 15.

18 Написати програму, що .зливає два відсортованих файла методом зовнішнього сортування.

19 Написати програму, що корегує записи файлу реєстратури з завдання 15. Корегування записів здійснювати за реєстраційним номером. Контролювати реєстраційний номер, котрий не повинен виходити за межовий реєстраційний номер.

20 Написати програму, яка сортує файл реєстратури з завдання 15 за прізвищем.

21 Написати програму, яка сортує файл з діагнозами із завдання 16. Сортування виконати за спеціалістами.

22 Написати програму, що стискає текст з файлу за наступним принципом: декілька, що ідуть підряд однакових символів замінити ознакою стискання, кількістю стиснутих символів, символом. Стискання виконувати лише у тому разі, якщо воно дає економію.

23 Написати програму, що розподіляє компоненти файлу реєстратури з завдання 15 на два файли, за ознакою чоловіки і жінки.

Література

Основна

1 "Руководство по GNU Emacs" Тринадцатая редакция, обновлено для Emacs версии 20.7 Ричард Столмен, Издано: АНО «ИЛКиРЛ» 1999г.

2 Хювенен Е. «Мир Лиспа. Введение в язык ЛИСП и функциональное программирование» (том 1) - М.: Мир. 1990. - 447с.

3 Хювенен Е. «Мир Лиспа. Введение в язык ЛИСП и функциональное программирование» (том 2) - М.: Мир. 1990. - 332с.

4 Peter Seibel "Practical Common Lisp" APRES. 2006г.

Додаткова

1 Хендерсон П. «Функциональное программирование. Использование и реализация». - М.: Мир. 1983. - 349 с.

А. Филд, П. Харрисон «Функциональное программирование»: Пер. с англ. - М.: Мир. 1993.-637с

Додаток А

Інтерпретатор і пакетна організація середовища Common Lisp

A.1 Склад дистрибутиву Practical Common Lisp

Для роботи з системою програмування Practical Common Lisp (CLisp-2.37) треба мати архів `lispbox-0.7-clisp-2.37` і розархівувати його у папку `lispbox-0.7-clisp-2.37`. Існують також інші версії Lispbox, наприклад `lispbox-0.7-clisp-1.6`.

Підчас розархівації утвориться папка `lispbox-0.7`, в яку входять папки:

- `clisp-2.37` з головною програмою `Lisp.exe`; сторінками доступу до документації через Інтернет; файлами необхідними для роботи системи;
- `emacs-21.3` з текстовим редактором Emacs;
- `portableaserve-20051204.214932` з адресами серверів та інформацією для рішення виниклих проблем;
- `slime-20060110` із оболонкою SLIME для взаємодії текстового редактора Emacs та Common Lisp;
- `practicals-1.0.3` – з кодом Practical Common Lisp.

У папку входить також файл `Lispbox.bat`. Файл `Lispbox` інсталує Common Lisp і одночасно запускає його.

A.2 Запуск Practical Common Lisp та вихід з нього

Для запуску Practical Common Lisp з текстовим редактором треба: увійти в каталог `lispbox-0.7-clisp-2.37/lispbox-0.7`; запустити командний файл Practical Common Lisp – `Lispbox.bat`.

Для виходу з Practical Common Lisp Можна вийти за командою меню FILE/ EXIT EMACS. Вийти тимчасово можна за командою “C-z”.

A.3 Робота з інтерпретатором

За підказкою інтерпретатору `CL USER>` можна ввести вираз.

Вирази, що поступають на вхід інтерпретатору, можуть бути: викликом функції; означенням функції користувача; даним,

командою, класом, методом. Розглянемо прості приклади роботи з інтерпретатором:

А.3.1 Вираз – виклик функції

Прикладом виразу - виклику стандартної функції, що додає два числа, є:

```
CL USER> (+ 3 6)
```

9

Інтерпретатор вводить вираз, утворює у середовищі Ліспу числа 3 і 6, якщо їх немає у середовищі, і обчислює вираз. Після чого виводить на екран результат роботи функції. Цей цикл можна повторювати. Обчислення виконуються *автоматично*, якщо обчислення не заборонено.

Виклик функції подають у формі списку в префіксальній формі. Список елементів списку записують у круглих дужках, а елемент від елемента відокремлюють пропуском.

А.3.2 Вираз – складне дане список

Прикладом виразу – складного даного є список чисел:

```
CL USER>'(2 4 6 8)
```

(2 4 6 8)

Знак „апостроф” перед списком означає заборону обчислення, інакше інтерпретатор шукатиме функцію 2. Знак „апостроф” означає скорочений запис функції QUOTE.

Записи: (QUOTE (2 4 6 8)) і '(2 4 6 8) тотожні.

А.3.3 Вираз - змінна

Важливим поняттям мови є символ. Символ має зовнішнє подавання і внутрішнє. Зовнішнє подавання символу може мати будь-який набір знаків. Внутрішнє подавання складається з покажчиків.

У мові Lisp змінну подають символом. Символом можна подати також ім'я функції. Застосування символу залежить від того, на що він містить покажчик. Якщо символ - має покажчик: на дане, то символом подають змінну. Привласнити дане змінній (створити із символу

показчик на дане) можна функцією Setq. Функція повертає дане. Наприклад:

```
CL USER> (Setq R '(1 4 3 8))
(1 4 3 8)
```

Функція Setq забороняє обчислювання значення свого першого аргументу - символу R. Одержати значення символу R можна подавши на вхід інтерпретатора вираз:

```
CL USER>R
(1 4 3 8)
```

Значення символу R обчислюється і повертається, якщо обчислення не забороняється. Змінна може не мати значення, тоді при звертанні до неї повертається повідомлення про помилку.

Наприклад, нехай змінній Count не привласнено значення:

```
CL USER>Count
```

Інтерпретатор повертає повідомлення:

```
EVAL: variable Count has no value
```

```
№ [ABORT] Return to SLIME top-level.
```

Для повернення до роботи треба встановити курсор на повідомлення і натиснути <Enter>.

A.3.4 Функція QUOTE

Можна заборонити обчислювання значення символу подібно тому, як це робить функція Setq.

Наприклад:

```
CL USER>'count
COUNT
```

При вводі символу буквами за замовченням буква нижнього регістру перетворюється у заголовну букву, якщо символ не взято у парні розділові знаки або перед нею не стоїть «\».

```
CL USER>'a
A
CL USER>'a
a
```

```
CL USER>'|a c|
|a c|
```

A.3.5 Вираз – означення функції

Прикладом виразу - означення функції користувача є функція, що знаходить суму першого і останнього елементів списку:

```
CL USER>(Defun sum (L) (+ (car L) (Car (Last L)) ) )
sum
```

Функція CAR бере перший елемент списку L і повертає його на місце її виклику. Функція LAST бере останній конс списку L, а функція CAR бере перший елемент консу і повертає його на місце виклику. Після чого функція „+” знаходить суму результатів роботи цих функцій.

Означення функції користувача компілюється в псевдокод. Якщо помилок немає, то повертається ім'я функції. Інакше видається повідомлення про помилку.

```
CL USER>(sum '(2 4 6 8) )
10
```

A.4 Середовище Common Lisp

A.4.1 Поняття середовища

СЕРЕДОВИЩЕ – це поточний стан оперативної пам'яті виділеної і структурованої певним чином. Середовище складається з усіх активних на даний момент структур: символів, констант, означень функцій, макросів.

A.4.2 Поняття пакету

Символи можуть об'єднуватися в пакети. З пакетами зручно працювати, якщо користувач бажає застосовувати певні бібліотеки. У програмі користувача і у бібліотеці можуть бути символи з однаковими іменами. Щоб уникнути помилок символи бібліотеки і символи програми користувача розміщують у іменовані пакети.

Перевірити чи існує пакет з вказаним ім'ям можна за функцією:

```
(Find – Package 'символ_ім'я_пакету)
```

```
CL USER> (Find – Package 'PR)
#<PACKAGE PR>
```

Функція повертає покажчик на пакет, якщо вказаний пакет є, інакше NIL. У пакеті символи розподіляються на зовнішні та внутрішні.

Внутрішні пакети інтерновані у ядро свого пакету. Зовнішні символи відносяться до інтерфейсу і доступні(експорт) іншим пакетам. Ядром кожного пакету є таблиця, компоненти якої подаються: « „Рядок з іменем” – Символ ».

Функції роботи з пакетами знаходять символ у пакеті за рядком-ім'ям. При вводі виразу інтерпретатор шукає у таблиці поточного пакету символ. Якщо символ не існує, то інтерпретатор утворює об'єкт - символ у середовищі, інакше символ не утворюється.

A.4.3 Створення пакету та його символів

Утворити новий пакет з вказаним ім'ям та списком псевдонімів можна за функцією `make-package`. Опція `:use` вказує на список імен пакетів, зовнішні символи яких будуть використовуватися у новому пакеті. Списки можуть бути відсутніми. *Функція повертає покажчик на новий пакет.*

```
(make-package "Рядок_ім'я_пакету" :nicknames список1 :use
список2)
```

Наприклад:

```
CL USER>(make-package "PR" : nicknames ('("pr") :use ('("PR2"))
#<PACKAGE PR>
```

Створення (інтернування) символу у пакеті виконується функцією:

```
(Intern "символ-рядок" 'Символ_ім'я_пакету)
```

Функція інтернує у пакет вказаний символ. Функція повертає складове ім'я символу.

```
CL USER> (intern "s" 'PR)
```

```
PR :: |s|
```

```
NIL
```

Де символ PR - ім'я пакету, розташоване перед двома джокрапками, а s символ. За замовченням поточний пакет NIL. Привласнити символу пакету значення можна:

```
CL USER> (Setq PR :: s 15)
15
```

Інтернування символу можна виконати у поточний пакет без вказівки пакету:

```
CL USER> (intern "s")
|s|
NIL
```

A.4.4 Робота з символами пакетів

До символів не з поточного пакету звертаються за складним іменем, яке містить дві двокрапки. Ім'я поточного пакету привласнено змінній **Package**. Перевірити чи входить символ у пакет можна за функцією:

```
(Find-symbol "символ-рядок" 'Символ_ім'я_пакету)

Наприклад: CL USER> (Find-symbol "s" 'PR )
PR::|s|
:INTERNAL (внутрішній символ)
```

Знайти всі пакети у які інтерновано вказаний символ можна за функцією:

```
(Find-all-symbols "символ у формі рядка" 'Символ_ім'я_пакету)

Наприклад: CL USER> (make-package 'p )
CL USER> (make-package 'a )
CL USER> (intern "C" 'p)
CL USER> (intern "C" 'a)
CL USER> (find-all-symbols 'c)
(p::|C| a::|C|)
```

До символів поточного пакету можна звертатися без вказівки імені пакету. Такі символи називають неспеціалізованими. Неспеціалізовані імена не містять двокрапок. Зробити пакет поточним можна:

```
CL-USER> ((in-package "R"))
#<PACKAGE R>
```

```
R> (in-package "COMMON-LISP-USER")
#<PACKAGE COMMON-LISP-USER>
CL-USER>
```

Розглянемо взаємодію між пакетами COMMON-LISP-USER і пакетом користувача L на прикладах.

Створимо пакет L і зробимо його поточним. Розмістимо у ньому функцію V, що вводить два числа та додає їх. Розмістимо також в пакеті змінну D і привласнимо їй значення.

```
CL-USER> (MAKE-PACKAGE 'L)
#<PACKAGE L>
CL-USER> (IN-PACKAGE "L")
#<PACKAGE L>
L> (DEFUN V (X Y)
(+ X Y))
V
L> (SETQ D 16)
16
```

Символи V і D локальні в поточному пакеті L. Для виклику функції V та змінної D з пакету CL-USER треба зробити символи V та D спеціалізованими зовнішніми у пакеті CL-USER, імпортуючи їх в цей пакет.

Після імпорту символу в пакет CL-USER він додається в таблицю символів цього пакету. Через імпорт символ можна зробити присутнім в багатьох пакетах.

```
L> (IMPORT '(L::V) 'CL-USER)
T
L> (IMPORT '(L::D) 'CL-USER)
T
```

Перейдемо в пакет CL-USER і одержимо значення змінної D та викличемо функцію V.

```
L> (IN-PACKAGE "CL-USER")
#<PACKAGE COMMON-LISP-USER>
CL-USER> (V 2 3)
5
```

```
CL-USER> D
```

```
16
```

До спеціалізованих зовнішніх символів із інших пакетів звертаються через одну двокрапку.

```
CL-USER> (MAKE-PACKAGE 'M)
```

```
#<PACKAGE M>
```

```
CL-USER> (IN-PACKAGE "M")
```

```
#<PACKAGE M>
```

```
M> CL-USER:D
```

```
16
```

Зробити символ зовнішнім, присутнім в обох пакетах можна через `export`.

```
CL-USER> (EXPORT '(CL-USER::D) 'L)
```

```
T
```

Тепер символ можна не тільки одержати з обох пакетів, а і змінювати в обох пакетах.

A.4.5 Стандартні пакети

Середовище містить 3 стандартних пакети: `Common-Lisp`, `Common-Lisp-user`, `Keyword`. При вході в `Common-Lisp` поточним пакетом є пакет `Common-Lisp-User`. Змінній `*Package*` привласнюється покажчик на цей пакет: `*#<PACKAGE COMMON-LISP-USER>`.

Пакет `Common-Lisp-User` застосовує пакет `Common-Lisp`, що експортує символи стандарту `Common-Lisp`. Користувачеві не дозволено інтернувати символи у пакет `Common-Lisp`. Пакет `Common-Lisp-User` є робочим пакетом. Користувач може інтернувати символи в цей пакет через `READ-EVAL-PRINT-LOOP` цикл `REPL`.

Пакет `Keyword` містить символи, що починаються з двокрапки. Вказані символи автоматично експортуються і мають покажчик на себе. Пакет `Keyword` дозволяє обмежити ключові слова, відкинути суперечливі ситуації у різних реалізаціях `Common-Lisp`.

Додаток Б

Робота у оболонці Slime

Б.1 Система програмування «Lisp in a Box»

Оболонка Slime – головна програма системи програмування. Вона доповнює меню текстового редактору Emacs опціями необхідними для роботи Common Lisp.

Всі дії у системі програмування можна виконувати опціями меню або командами з клавіатури.

Оболонка Slime дозволяє виконувати програми, які були створені на різних реалізаціях Common Lisp з використанням стандарту, згладжуючи розбіжності у кодах.

При помилках у програмі користувача можлива ситуація, коли розірвано зв'язок Common Lisp з оболонкою Slime. Для поновлення зв'язку застосуйте команду: ALT – X SLIME.

Текстовий редактор Emacs має всі властивості сучасних розвинених редакторів: розширюватись, користувач може додавати нові команди написані на мові Лісп; налаштовуватись - можна призначати команди до інших клавіш.

Редактор змінює меню, залежно від ситуації; працює в багатозадачному режимі(під час виконання команди можна редагувати текст).

Б.2 Організація X - вікон та буферів

Б.2.1 Поняття буфера та X – вікна

При вході в систему програмування «Lisp in a Box» активним стає буфер, що взаємодіє з Common Lisp.

Буфер це – об'єкт редактору. Через буфер редактор взаємодіє з файлами, користувачем. У буферах редактор зберігає текст.

Кожному буферу на екрані відповідає одно або більше вікон. Вікна текстового редактору Emacs і екран називають X - вікнами. X - вікно або екран є подається фреймом(структурою). Фрейм – екран містить рядок – меню та один або декілька фреймів (X-вікон). Фрейм X-вікно складається з середньої робочої частини; під нею рядку

режиму роботи з буфером, нижче області луни. У області луни відображаються підказки користувачу, відлунюються події у системі.

При вході в Emacs утворюються стандартні буфери необхідні для роботи системи:

1) Буфер **slime-repl clisp-2.37** використовується для роботи Common Lisp. Режим роботи буфера - опція меню REPL, визначає команди керування роботою Common Lisp.

2) Буфер **scratch** застосовуються для зберігання текстів; може використовуватись для обчислення виразів у текстовому редакторі Emacs.

3) Буфер **Messages** використовується для зберігання та накопичення повідомлень про події, що відбуваються. У буфері відображаються всі повідомлення системи про результати дій під час завантаження системи.

4) Буфер **inferior-lisp** використовується для повідомлення користувачу про результати роботи з Lisp на нижньому рівні.

5) Буфер **Compile-Log** містить журнал компіляції.

6) Буфер **Slime-events** містить відомості про результати дій у оболонці Slime.

У певний момент кожен стандартний буфер може працювати у одному *основному режимі*: Fundamental, Text, Lisp, C, Texinfo та інші.

Б.2.2 Створення та закриття буферів

Під час роботи користувача можуть з'являтися інші стандартні буфери або буфери користувача:

1) При відкритті файлу(опція меню Open File) для його вмісту автоматично створюється буфер з ім'ям файлу та X – вікно. Ім'я буферу може бути будь-якої довжини.

2) Для вводу параметрів команд Emacs автоматично створюється міні буфер у області луни. Наприклад, за командою Open Directory в області луни створюється міні буфер для вводу імені каталогу, а для відображення вмісту каталогу утворюється буфер з ім'ям каталогу. Додаткова інформація міні буфера використовується функціями Lisp як фактичні параметри.

3) Для вмісту довідки або з іншими цілями може динамічно утворюватись буфер. Наприклад, буфер HELP.

4) Користувач може утворити порожній буфер з будь-яким ім'ям за командою C-x b. Ім'я буферу опитується у міні буферу. Вийти з міні буферу можна за командою C-g.

Закрити буфер на весь сеанс роботи з Emacs можна за командою Close(current buffer). Згорнувши вікно, можна тимчасово закрити буфер або поставити у відповідність X- вікну інший буфер.

Б.2.3 Створення X - вікон

Вікна можуть створюватись автоматично або за командою (опцією меню) користувача:

1) Вікна автоматично створюються для вмісту файлів, каталогів або параметрів команд.

2) Редактор запускається з одним X-вікном, але можна утворити ще вікно за опцією меню *File/ Split Window* або командою C-x 2. Опція ділить активне вікно горизонтальною лінією. Активним називають вікно, у якому знаходиться кольоровий курсор.

3) Створити нове вікно можна командою C-x 3. Опція ділить активне вікно вертикальною лінією.

4) Зробити одне вікно на екрані для обраного буферу командою (C-x 1) незалежно від того, скільки вікон було утворено на екрані.

Б.2.4 Довідкова інформація про буфери

Позиція курсору в тексті буфера відмічається кольоровим прямокутником(точкою). Щоб запам'ятати позицію курсору у тексті буферу, що був обраним раніше, використовується порожній прямокутник(точка).

Більшість команд та увід відноситься до обраного буферу. *Команди миші відносяться до буферу на якому виконано щиглик.*

При роботі з текстом у буфері використовуються символи таблиці ASCII від 0...126 у десятковій системі. Можливо також використовувати символи, що друкуються і не входять в таблицю ASCII і кожен з них подається двома та більше байтів, якщо включена можливість багатобайтні символи.

Рядок режиму буфера

Кожен буфер має рядок режиму. Щоб показати рядок режиму для вікна, треба розгорнути вікна на екрані у ширину.

Рядок режиму містить назву буфера. Для швидкої зміни у вікні буферу треба клацнути на назві буферу у рядку режиму. Структуру рядка режиму дивися нижче.

Рядок містить: 5 символів, Назву буфера, Основний режим, Другорядні режими, Номер поточного рядку, Позицію.

Перший символ

Встановлюючи курсор на 1-ий символ рядку, можна одержати інформацію про можливі у даному контексті команди:

Другий символ

Встановлюючи курсор на 2 символ рядку, можна одержати інформацію про набір кодування системи, а натиснувши праву клавішу миші можна одержати буфер з повним описом кодування: 1 - кодування ISO Latin 1; 5 - кодування Cyrillic 5. Якщо 2 символ містить: «-», то кодування по замовчанню. Щоб перевірити вказане Завантажте файл у буфер.

Третій символ

За вибором методу вводу до символу 3 додається «метод». Метод вказує, як закінчується рядок. Символ 3 заміщується на « \ », якщо ознака кінця рядку «BK» та «PC». Іноді «:» заміщується на «UNIX».

Четвертий та п'ятий символи

Символи 4,5 подаються «**», якщо буфер вікна редагувався. Якщо буфер вікна не редагувався подаються «--». Встановлюючи курсор на символ 4, можна одержати режим роботи з буфером: «тільки читати» або «читати-писати».

Знак «%» у 4 символі означає, що режим «тільки читати». Знак «*» означає, що режим «читати – писати». Встановлюючи курсор на другу «*», можна одержати інформацію про команду модифікування режиму та за правою кнопкою миші змінити вказаний режим.

Основні та другорядні режими

У дужках рядка показано назву основного і другорядних режимів роботи з буфером. Наприклад, при роботі з Common LLisp

використовується основний режим - REPL, тобто у буфері можна застосовувати команди опції REPL.

Другорядний режим це – режим, який включено для обраного буфера і тільки для обраного вікна. Обрати другорядний режим можна встановивши покажчик миші на основний режим у рядку режиму та натиснувши праву кнопку миші.

Прикладом другорядних режимів можуть бути Overwrite та Abbrev. Перший режим дозволяє перезаписувати файл, а другий працювати з командами редагування тексту, які вживає текстовий редактор WORD.

Інформація про текст

У рядку біля літери «L» вказано номер рядка, у якому знаходиться курсор. Позиція All вказує, що у вікні показано всі рядки буфера; символ TOP вказує, що курсор у першому рядку; символ BOT вказує, що курсор у останньому рядку, «число %» показує скільки % тексту знаходиться зверху вікна.

Область луни

Область луни може відлунювати знаки, що набирає користувач; містити команди, що необхідні користувачу у певний час; інформацію про помилки при роботі; інформацію про виконання команди, або при зупинці підказує, що треба вводити далі. Інформаційні повідомлення з області луни накопичуються у буфері *Messages*.

Міні буфер

Якщо команда або опція вимагає додаткової інформації, то автоматично в області луни з'являється міні буфер і він стає поточним буфером.

Міні буфер завжди підказує, що повинен ввести користувач. Підказка закінчується символом двокрапка. Вона може містити замовчення для відповіді. Користувач вводить інформацію та закінчує увід „ENTER”.

Для допомоги застосовують команду C-j, у новому вікні буде підказка, що може ввести користувач.

Можна ввести у міні буфер перші символи команди і завершити їх пропуском або „ENTER”. Редактор EMACS запропонує можливі продовження. Можна користуватись командою авто доповнення:

ввести один або декілька букв і натиснути клавішу TAB. У вікно HELP виводиться список імен, що починаються з цих букв.

Якщо є один варіант, то натиснувши клавішу TAB, команда поповниться автоматично. При багатьох варіантах оберіть правою кнопкою миші один варіант та натисніть „ENTER”, команда поповниться та виконається.

При роботі з міні буфером можуть виникнути ситуації, коли міні буфер потрібен для інших цілей. Тоді, міні буфер зберігається, а після вводу користувача повертається.

Усі команди введені у міні буфер зберігаються в списку – історія. Список можна переглянути рухаючись за стрілками: вгору, вниз. Команди можна використовувати повторно.

Для скасування команди, яку ви викликаєте, виконайте команду C-g. За командою міні буфер не активний і виконується перехід на верхній рівень.

Б.3 Опції текстового редактору

Б.3.1 Меню текстового редактору

Оболонка текстового редактору EMACS має головне меню, опції якого призначено для роботи з текстовим редактором EMACS, системою програмування Common Lisp, оболонкою SLIME.

Головне меню складається з опцій: FILE, EDIT, OPTIONS, BUFFERS, TOOLS, SLIME, REPL, HELP.

Опція FILE:

– Open File... Відкривається поточний каталог, щоб обрати і відкрити файл, після чого файл розміщується у буфері з ім'ям файлу. Треба пам'ятати, що всі опції які вимагають додаткових даних від користувача, закінчуються трьома крапками «...»;

– Open Directory... За замовченням опція виводить у міні буфер шлях до поточного каталогу. Шлях можна корегувати, або вилучити повністю і ввести. За „Enter” у вікно виведеться вміст каталогу. Обрати ім'я файлу кнопкою миші та натиснути „Enter”. Вміст файлу завантажуватиметься у буфер.

– Insert File... Відкривається поточний каталог, обрати і завантажити файл з програмою на вхід інтерпретатору Common Lisp.

Натиснути <Enter>. Програма виконується. Повертається результат останньої виконаної дії;

- Close (current buffer) Опція закриває поточний буфер. Буфер не буде доступним під час сеансу Common Lisp, що відбувається;
- Save (current buffer) Зберегти вміст поточного буфера в файлі з ім'ям буфера. Якщо ім'я файлу змінити, то змінюється ім'я буфера;
- Save buffer as...Зберегти вміст буфера у файлі. Якщо файл з вказаним ім'ям існує, то запитується, чи треба переписати його, інакше користувач вводить нове ім'я і змінюється ім'я буфера;
- Split Window Зробити копію вікна для того ж буфера, розділивши вікно горизонтальною лінією;
- Unsplit Windows Згорнути всі вікна у одне;
- New Frame Запустити ще одну копію системи Common Lisp;
- Delete Frame Вилучити копію системи Common Lisp;
- Exit Emacs Вийти з системи.

Опція EDIT

Опція дозволяє копіювати, вставляти, вилучати текст буфера, рухатись по тексті, знаходити текст або замінювати його. Опції копіювання, вибору, вставки та вилучання тексту виконуються звичайним способом.

Опції пошуку або заміни в тексті фрагменту тексту застосовують міні буфер для вводу фрагментів тексту. Після знаходження фрагменту тексту, курсор встановлюється на кінці знайденого тексту при пошуку вперед і на початку знайденого тексту при пошуку назад.

Команди групи Advanced Search / Replace дозволяють виконувати пошук та заміну виразів, що часто зустрічаються (регулярних виразів), або дозволяють виконувати пошук у всіх файлах за ознакою(Search in all tagged files), у іншому вікні за ознакою (Find Tag In Other Window...).

Команда Fill дозволяє заповнити інформацією рядок на всю ширину.

Команда Text Properties дозволяє: змінювати колір тексту, тип шрифту; встановлювати та вилучати режим роботи з текстом (наприклад, тільки читати); вирівнювати або центрувати текст, робити відступ; одержувати список всіх властивостей тексту.

Опція OPTIONS

Опція встановлює додаткові можливості для роботи з текстовим редактором та Common Lisp:

- Syntax Highlighting (Global Font Lock mode) - Виявляє місце з синтаксичною помилкою.
- Paren Match Highlighting(Show Paren Mode) – Підсвічує парні дужки.
- Truncate Long Lines in this Buffer – розбиває рядок на два і виводить маркери стрілки на екран, за якими можна швидко переглянути рядок.
- Word Wrap in Text Modes (Auto Fill) – дозволяє застосовувати команди текстового редактору Word.
- Enter Debugger on Error – Увійти у налагоджувач за помилкою.
- Enter Debugger on Quit - Вийти з налагоджувача.
- Mule (Multilingual Environment) МЕНЮ Mule багатомовного навколишнього середовища.
- Set Language Environment - Встановити мовне середовище.
- Set Font / Font set Встановити шрифт/ набір символів.
- Toggle Input Method Переключити метод вводу.
- Select Input Method Обрати метод вводу.
- Set Coding Systems МЕНЮ Встановити кодування системи.
- Save Options Зберегти опції.
- Customize Emacs -> МЕНЮ Налаштування Emacs.

Опція BUFFERS:

Опція дозволяє призначати для вікна з поточним буфером інший буфер або одержати список буферів, існуючих у системі.

Опція TOOLS

- Опція забезпечує користувача інструментальними засобами для пошуку файлів; налагоджування функцій редактору; порівняння,

злиття файлів; налагодження файлів або буферів; спілкування у Інтернеті, тощо.

- Compile...дозволяє компілювати файл з програмою, а потім завантажити його командою Load та виконати. Компілювати файл можна функцією (compile-File "name"). Відкомпільована програма має розширення ".fas". Функція повертає покажчик на файл.

- Display Speed bar – дозволяє швидко переглядати вміст каталогів та рухатись по каталогах.

- Directory Search – дозволяє зв'язатись з серверами з яких можна розширити інформацію необхідну для роботи Common Lisp.

Опція SLIME

Опція Documentation дозволяє переглянути у Інтернеті документацію з функціями мови Common Lisp Hyper Spec, яку використовує Практичний Лісп. Команди опції дозволяють виконувати дії над функціями оболонки Slime.

Опція REPL:

- Встановити курсор на початок функцій, які виконувались раніше і треба виконати знову. Натиснути клавішу <Enter>. Функції переносяться на вхід інтерпретатора Common Lisp. Функції можна корегувати без буферу. Теж саме виконує опція Send Input.

- Interrupt Lisp process (C-c C-c) Перервати зациклення програми.

- Previous Input (M-p) Переносить на місце вказане курсором попередній вираз. Функція або виклик функції, що були на цьому місці заміщуються.

- Next Input (M-n) Переносить на місце вказане курсором наступний вираз. Функція або виклик функції, що були на цьому місці заміщуються.

- Goto Previous Prompt (C-c C-p) Перенести курсор на попередній вираз.

- Goto Next Promt (C-c C-n) Перенести курсор на наступний вираз.

- Clear Last Output (C-c C-o) Встановити курсор на вираз, який треба вилучити. За починаючи з цього виразу всі вирази очищаються. Перший вираз не очищається.

- Clear Buffer (C-c C-t) Очищає буфер повністю. В середовищі Common Lisp всі попередньо встановленні змінні та відкомпільовані у Дкод функції зберігаються.

- Kill Current Input (C-c C-u) Очищає вхідний потік.

Опція HELP:

- Опція Manuals дозволяє одержати довідку про роботу з текстовим редактором.

- Опція Describe описує функції, змінні, команди за ім'ям.

- Опція Find Emacs Packages – вказує ключові слова для пошуку у бібліотеках Emacs.

Б.4 Команди текстового редактору з клавіатури

Загальні відомості про команди з клавіатури

Виконати команду можна увівши її з клавіатури. При вивченні команд потрібні наступні відомості. Поточними каталогом вважається каталог, що має Lispbox.bat. Для роботи з командами необхідно встановити нижній регістр англійського алфавіту.

Символ “C” означає клавішу CTRL. Символу “M” відповідає клавіша ALT. Тире “-” означає, що натиснуту першу клавішу, треба держати одночасно з наступною. Наприклад, C-h. Клавішу символу, який зображено через пропуск, натискають окремо. Наприклад, C-h w.

Послідовність набраних символів поєднаних рискою означає одну подію. Символ, який набрано окремо, означає другу подію.

Таким чином, команда „C-h w” – означає дві події: звернутись до довідки і знайти де у меню розташована команда, ім'я якої вказується у міні буфері.

Послідовність подій називають ключем. Ключ може бути закінченим – послідовність подій достатня для опису команди. Увід закінченого ключа - запускає команду.

Ключ може бути префіксальним. Префіксальний ключ вказує на групу команд. Продовження префіксального ключа може зробити ключ закінченим. Обмеження на довжину ключа немає.

Користувач може утворювати нові префіксальні ключі, використовуючи стандартну прив'язку або вилучати старі ключі. Вилучення ключів впливає на команди групи, тому вилучення треба робити обережно.

Текстовий редактор має таблиці ключів. Кожна компонента таблиці встановлює зв'язок між ключем(командою користувача) і внутрішньою командою(функцією на мові EMACS LISP). Функція одержує від користувача необхідні дані та передає їх програмі для виконання дій. Програма, що виконує дії також написана на мові LISP.

Команди роботи з довідкою

Група команд визначається префіксальною командою C-h або F1.

- 1)Вивести EMACS документацію: C-h t.
- 2)Вивести зміст EMACS документації: C-h i.
- 3)Вивести повну довідку за командою: C-h ?. Вказавши у міні буфері опцію відбору команд можна одержати довідку про групу команд; одержати відомості про ключі; кодування системи; функції, змінні Common Lisp, тощо.
- 4)Знайти за ключем опис команди: C-h k. Вводиться команда та ім'я ключа в міні буфер. Команда дає повну інформацію про ключ. Наприклад, ключ C-x C-f..

Команди роботи з файлами

Текст буферу з вже обраним ім'ям зберігається у файлі за командою C-x C-s.

Завантажити файл в буфер текстового редактора можна за командою C-x C-f. У міні буфер виводиться ім'я поточного каталогу та пропонується ввести ім'я файлу. Наявність файлу викликає завантаження файлу його вмісту в буфер для редагування або друку.

Б.5 Робота системою програмування

- 1) Завантажити файл в буфер текстового редактора можна за командою FILE/OPEN FILE.

Текстовий редактор EMACS дозволяє працювати не тільки з файлами, що підготовлені на ньому, а і файлами, що підготовлені у блокноті. Корегувати програму можна на вході інтерпретатору.

2) Після корегування файлу його зберігають у файл.

3) Завантажити програму для виконання інтерпретатором Common Lisp із файлу, треба виконати опцію FILE/INPUT FILE.

4) Для виконання програми натиснути <ENTER>. При повідомленні про помилку функцією EVAL прочитати повідомлення з Restarts і обрати опцію для повернення у Common Lisp.

Рекомендується обрати опцію „Return to SLIME top-level”, щоб повернутись у оболонку SLIME на верхній рівень. Для цього треба встановити курсор на номер опції і натиснути <ENTER>.

5) Для швидкого завантаження програми її треба попередньо відкомпілювати функцією: (compile-file „ім'я файлу”), а потім завантажити функцією: (Load “ім'я файлу.fas”).

Треба пам'ятати, що виконана компіляція є компіляцією в псевдокод для інтерпретатора.

Додаток В

Базові функції

В.1 Базові функції відбору

Функції розділу працюють тільки зі списками та консами (точковими парами). У інших випадках інтерпретатор фіксує помилку. Функції виконують дії не змінюючи список або конс.

Функція CAR - повертає перший елемент списку або точкової пари. Наприклад:

```
CL-USER> (CAR '(2 4 6))
2
CL-USER> (CAR '(5 . 7))
5
```

Функція CDR - повертає хвіст списку або другий елемент точкової пари. Наприклад:

```
CL-USER> (CDR '(2 4 6))
(4 6)
CL-USER> (CDR '(5 . 7))
7
```

Складні функції – є комбінацією функцій CAR і CDR до 4-х рівнів вкладеності: C.....R.

Складна функція повертає результат роботи комбінації функцій. Наприклад:

```
CL-USER>(CADR '(2 4 6))
4
Вираз (CADR '(2 4 6)) тотожний виразу (CAR (CDR '(2 4 6))).
CL-USER> (CDDR '(5 7 24 17))
(24 17)
CL-USER> (CADDR '(2 4 6 8 9))
8
CL-USER>(CDAR '((5 7) (2 3)) )
(7)
```

В.2 Базові функції конструктори

Об'єктами функцій можуть бути об'єкти будь-яких типів.

Функція *CONS* – будує нову точкову пару, перший елемент якої містить покажчик на перший аргумент функції *CONS*, а другий елемент пари містить покажчик на другий аргумент функції *CONS*. Функція повертає одержаний об'єкт. Наприклад:

```
CL-USER> (CONS 10 6)
(10 . 6)
CL-USER> (CONS 10 '(2 4 6))
(10 2 4 6)
CL-USER> (CONS '(3 6 7) '(2 4 6))
((3 6 7) 2 4 6)
CL-USER> (CONS 10 NIL)
(10)
```

Функція *LIST* – будує список з своїх об'єктів і повертає його. Функція може мати скільки-завгодно аргументів.

```
CL-USER> (LIST 5 8 9)
(5 8 9)
CL-USER> (SETQ N 3 M 7 L 6) (LIST N M L)
( 3 7 6)
CL-USER> (LIST '(1 2) '(3 4) '(5 6) )
(( 1 2) (3 4) (5 6) )
```

В.3 Функції компаратору

Функції мають аргументами два об'єкти. Функції порівнюють об'єкти і повертають істину чи хибність. Кожна функція компаратору має свою умову за якою виконується порівняння об'єктів.

Функція *EQ* – перевіряє, чи знаходяться об'єкти у середовищі за однією адресою(перевірка на тотожність об'єктів).

Об'єктами можуть бути: знаки, символи, цілі числа, що розміщуються у 3-х байтах($N < 2^{24}$), логічні константи(*T*, *NIL*). Функція

не порівнює рядки, списки, конси, числа різних типів та числа, що подані у різних формах. Для таких об'єктів функція повертає NIL.

```
CL-USER> (SETQ A #\w) (EQ #\w A)
```

```
T
```

```
CL-USER> (EQ 24 24)
```

```
T
```

```
CL-USER> (EQ 'a 'A)
```

```
T
```

```
CL-USER> (EQ NIL '())
```

```
T
```

```
CL-USER> (EQ '(1 4) '(1 4))
```

```
NIL
```

```
CL-USER> (EQ 16777216 16777216)
```

```
NIL
```

```
CL-USER>(EQ 16777215 16777215)
```

```
T
```

```
CL-USER> (EQ 2.0 2)
```

```
NIL
```

```
CL-USER> (EQ 20.0 8/4)
```

```
NIL
```

Функція *EQL* – порівнює два об'єкти за величиною. Обмежень на числа нема. Об'єктами функції не можуть бути списки, конси, інші структури. Порівняння вказаних об'єктів повертає NIL. Всі інші об'єкти порівнюються.

```
CL-USER> (EQL 16777216 16777216)
```

```
T
```

```
CL-USER> (EQL '(2 4) '(2 4))
```

```
NIL
```

```
CL-USER> (SETQ A 2 B 2) (EQL A B)
```

```
T
```

```
CL-USER> (EQL 4/2 2)
```

```
T
```

```
CL-USER> (EQL 0.5 5e-1)
```

```
T
```

Функція *EQUAL* – порівнює списки, точкові пари, будь-які структури. Функція не порівнює числа різних типів та числа, що подані у різних формах.

```
CL-USER> (EQUAL '(1 2 3) '(1 2 3))
```

```
T
```

```
CL-USER> (EQUAL '(2 . 4) '(2 . 4))
```

```
T
```

```
CL-USER> (EQUAL '(2.0 5) '(2 5))
```

```
NIL
```

Функція *=* – Функції порівнює тільки числа. Якщо аргументи функції не числа, то помилка. Функція може порівнювати числа різних типів та числа, що подаються у різних формах.

```
CL-USER> (= 3.0 3)
```

```
T
```

```
CL-USER> (= 0.5e+1 5.0)
```

```
T
```

```
CL-USER> (= 'B 'b)
```

Повідомлення „B не число”.

```
CL-USER> (setq H 4 r 4) (= H r)
```

```
T
```

Функція *EQUALP* – універсальна, порівнює будь-які дані з будь-якими аргументами.

```
CL-USER> (EQUALP '(1 2.0 3) '(1 2 3))
```

```
T
```

```
CL-USER> (EQUALP '(2 . 4.3) '(2 . 0.43e+1))
```

```
T
```

В.4 Функції предикати

Кожна функція має аргументом один об'єкт і перевіряє його тип.

Функція *NUMBERP* – перевіряє чи є об'єкт числом.

```
CL-USER> (SETQ M 12)(NUMBERP M)
```

```
T
```

```
CL-USER> (NUMBERP 'C)
```

NIL

Функція INTEGERP – перевіряє, чи є об'єкт цілим числом.

CL-USER> (SETQ M 12)(INTEGERP M)

T

CL-USER> (INTEGERP 0.5)

NIL

Функція PLUSP – перевіряє, чи є об'єкт додатнім числом.

CL-USER> (SETQ M 12)(PLUSP M)

T

CL-USER> (SETQ M -12)(PLUSP M)

NIL

CL-USER> (PLUSP 'C)

Повідомлення „Не числовий аргумент”

Функція MINUSP – перевіряє, чи є об'єкт від'ємним числом.

CL-USER> (SETQ M -12)(MINUSP M)

T

CL-USER> (SETQ M 12)(MINUSP M)

NIL

CL-USER> (MINUSP 'C)

Повідомлення „Нечисловий аргумент”

Функція ZEROP – перевіряє, чи є об'єкт нулем.

CL-USER> (SETQ M 0)(ZEROP M)

T

CL-USER> (SETQ M 12)(ZEROP M)

NIL

Функція ODDP – перевіряє, чи є об'єкт непарним числом.

CL-USER> (SETQ M 11)(ODDP M)

T

CL-USER> (ODDP 12)

NIL

Функція EVENP – перевіряє, чи є об'єкт парним числом.

CL-USER> (SETQ M 12)(EVENP M)

T

```
CL-USER> (EVENP 11)
```

```
NIL
```

Функція *ATOM* – перевіряє, чи є об'єкт атомом (числом будь-якого типу, символом, рядком).

```
CL-USER> (SETQ M "as")(ATOM M)
```

```
T
```

```
CL-USER> (ATOM '(1 2 3))
```

```
NIL
```

Функція *LISTP* – перевіряє, чи є об'єкт списком.

```
CL-USER> (SETQ M '(2 3))(LISTP M)
```

```
T
```

```
CL-USER> (LISTP 'M)
```

```
NIL
```

Функція *CONSP* - перевіряє, чи є об'єкт консом.

```
CL-USER> (SETQ M '(A . B))(CONSP M)
```

```
T
```

```
CL-USER> (CONSP 'C)
```

```
NIL
```

Функція *NULL* перевіряє, чи є об'єкт порожнім списком або логічною константою *NIL*.

```
CL-USER> (SETQ M NIL)(NULL M)
```

```
T
```

```
CL-USER> (NULL '(1 2))
```

```
NIL
```

В.5 Керуючі спеціальні форми роботи із контекстом

Спеціальна форма *QUOTE* – забороняє обчислення свого аргументу. Скорочене позначення спеціальної форми *QUOTE* – одиночна лапка(').

У функції *Setq* кінцевий символ „q” означає заперечення обчислення:

```
CL-USER> (SETQ H 23)
```

```
CL-USER> H
```

```
23
```

```
CL-USER>'H
H
```

В.6 Керуючі спеціальні форми послідовного виконання

Спеціальна форма PROGN: - має аргументами форми і організує їх послідовне виконання. Спеціальна форма повертає результат останньої виконаної форми.

```
CL-USER> (PROGN (Setq M 3)(Print (+ 2 M)) (setq N 15))
5
15
```

Спеціальна форма PROGI: - має аргументами форми і організує їх послідовне виконання. Спеціальна форма повертає результат першої виконаної форми.

```
CL-USER> (PROGN (Print (+ 2 3)) (setq N 15))
5
```

В.7 Керуючі спеціальні форми розгалуження обчислень

Спеціальна форма IF – залежно від істинності умови обчислює THEN- форму або ELSE-форму:

(IF умова THEN-форма ELSE-форма)

Слова THEN та ELSE не пишуться. Умовою може бути будь-яка форма: константа, змінна, виклик функції. У мові Лісп істинним є будь-яке значення крім константи NIL.

Формою THEN або ELSE може бути тільки одна форма. Щоб виконати декілька форм застосовують спеціальну форму PROGN.

Наприклад:

```
CL-USER> (DEFUN Sp () (IF (ZEROP (READ))
  (PROGN (PRINT "Нуль") (setq N 1) )
  (PROGN (PRINT "Не нуль" ) (setq N 0) ) ) )
SP
```

```
CL-USER> (sp)
```

```
Sp
```

```
Увід: 25
```

Результат: ; Пропуск рядку

“НЕ нуль” ; Друкує Print.

0 ; Повертає функція Sp.

Спеціальна форма COND – *Спеціальна форма* має у тілі пари. Кожна пара визначає гілку програми і містить умову та дію. Пари переглядаються зліва направо. Якщо умова пари істина, то виконується дія пари. Результат дії повертається. Якщо жодна умова неістинна, то спеціальна форма повертає NIL. У разі потреби виконати декілька дій застосовують PROG.

Спеціальна форма *COND* дозволяє розгалужувати програму на багато гілок. При програмуванні першими гілками ставлять гілки із спеціальними умовами. Остання гілка - пастка з умовою „Т” для всіх випадків, що залишилися.

Спеціальна форма *COND* завершує виконання циклу, рекурсії, програми. Наприклад: Функція вводить число і залежно від цього друкує повідомлення.

```
CL-USER> (DEFUN Pr1 () (SETQ B (READ))
(COND ((PLUSP B) (PRINT “Число додатне”))
((MINUSP B) (PRINT “Число від’ємне”))
(T (PRINT “НУЛЬ”)) )..)
```

Pr1

CL-USER>(Pr1)

Увід:-14

Результат: „Число від’ємне”; Друкується функцією Print.

„Число від’ємне”; Повертається функцією С.

Додаток Г

Рекурсія

Г.1 Поняття рекурсії

Дії, що повторюються зручно виконувати рекурсією. Рекурсія компактна, зберігає дані для всіх ітерацій у стеку.

Рекурсивна функція має в своєму тілі виклик себе, але з іншими фактичними параметрами. Щоб такий процес приводив до результату, його треба обмежити граничною умовою, а фактичні параметри рекурсивного виклику повинні сходитись до відповідних значень у граничній умові.

Прикладом математичного рекурсивного означення є означення факторіала. Воно подане нижче:

$n! = n * (n - 1)!$ – рекурсивна умова.

$0! = 1$ – гранична умова.

Рекурсивною функцією зручно працювати з рекурсивними даними. Наприклад, списками, рядками, деревами. Рекурсивною структурою називають дане частина якого того ж самого типу, що дане.

Існують два типи рекурсивних методів організації функцій: низхідний і висхідний.

Г.1.1 Низхідний метод рекурсії

При низхідному методі рекурсії існує дві фази:

- 1) Фаза редукції.
- 2) Фаза одержання рішення.

На фазі редукції функція викликає саму себе, а аргументи функції змінюються послідовно до тих пір, поки не буде задовольнятися гранична умова.

При виконанні граничної умови функція повертає тривіальне(просте) рішення.

На фазі одержання рішення з тривіального рішення поступово одержують рішення задачі.

Демо рекурсивне означення функції „Факторіал” низхідним методом:

```
CL-USER> ( DEFUN FACT ( N )
  (COND ( (ZEROP N ) 1 )
    (T (* N ( FACT ( - N 1) ) ) ) )
  FACT
```

```
CL-USER> (FACT 4)
```

Результат:24

У нашому прикладі при граничній умові: $N = 0$, функція FACT повертає 1. Тому при рекурсивному виклику N зменшується поки не стане дорівнювати 0 (Фаза редукції).

Після чого виконується фаза „Отримання розв’язку”. Попередній результат для факторіала (N-1) множиться на число N. Дії виконуються до тих пір, поки не буде одержано розв’язок самої задачі.

Г.1.2 Висхідна рекурсія

Висхідна рекурсія відрізняється тим, що розв’язок одержується відразу ж на першій фазі – фазі одержання рішення.

```
CL-USER> (DEFUN FACT ( K F N)
  ( COND (( EQL K N ) F )
    (T ( FACT ( + K 1 ) (* ( + K 1 ) F ) N ) ) ) )
```

, де K поточне число, N граничне число, F факторіал числа

```
CL-USER> (FACT 0 1 4)
```

24

Додаток Д

Рекурсивна обробка списків

Від вибору типу рекурсії залежить простота та компактність програми

Розглянемо завдання. Дан список чисел. Написати програму, що повертає список, елементи якого записані у зворотному порядку. Дамо означення функції написаної висхідним методом рекурсії.

```
CL-USER> (DEFUN REV (L1 L2)
            (COND ((NULL L1) L2)
                  (T (REV (CDR L1) (CONS (CAR L1) L2))))))
```

REV

```
CL-USER> (REV '(1 2 3) NIL)
(3 2 1)
```

Функція REV при кожному рекурсивному виклику зменшує на один елемент список L1 і переносить його на початок списку L2. При такому приєднанні елементу до списку L2 елементи списку записуються у зворотному порядку.

Якщо використовувати низхідний метод рекурсії, то функція CONS буде виконуватись після досягнення граничної умови і тому елементи списку запишуться у тому ж порядку. Тобто ми одержимо копію списку.

```
CL-USER> (DEFUN REV (L1 )
            (COND ((NULL L1)NIL )
                  (T (CONS (CAR L1) (REV (CDR L1) ) ) ) ) )
```

REV

```
CL-USER> (REV '(1 2 3))
(1 2 3)
```

Щоб написати програму низхідним методом нам прийдеться ускладнити її. Для цього зробимо з першого елементу списку L1 список і об'єднаємо функцією APPEND два списки так, щоб перший елемент списку L1 записувався у кінець списку результату.

```
CL-USER> (DEFUN REV (L1 )
            (COND ((NULL L1) NIL )
                  (T (APPEND (REV (CDR L1)) (LIST (CAR L1))))))
```

Додаток Е

Спеціальні форми організації циклів

Для виконання дій, що повторюються, існують спеціальні форми.

Е.1 Спеціальна форма DO

(DO ((Змінна Значення Крок) ... (Змінна Значення Крок))
(умова_закінчення результат) Форма1 ... ФормаN)

Перед початком роботи спеціальної форми DO локальним змінним надають початкові значення, що будуть застосовуватись у циклі при обчисленні форм. Крок може не вказуватись. Тоді на кожній ітерації програміст сам повинен змінювати значення змінних. Після закінчення роботи циклу локальні змінні знищуються.

DO організує повторне обчислення форм, що розташовані в тілі DO зліва направо. Перша форма є умовою закінчення роботи циклу. Коли умова закінчення циклу стане істиною, тоді повертається результат. Інакше обчислюються наступні форми.

Розглянемо приклад1. Ввести числовий список. Одержати суму елементів списку.

```
CL-USER> (DEFUN SUM (L)
            (DO ((S 0))
                ((NULL L) S)
                (SETQ S (+ S (CAR L)) L (CDR L))))
CL-USER> (SUM '(1 2 3 4))
10
```

У виразі ((NULL L) S) дві дужки підряд означають неявну форму COND. Запис тотожній такому: (COND ((NULL L) S))

Якщо при виклику функції треба зробити кількість фактичних параметрів менше кількості формальних параметрів, то формальним параметрам, що не одержують значення привласнюється значення у списку локальних змінних.

Приклад2: Ввести числовий список. Утворити новий список, записавши елементи введеного списку у зворотному порядку.

```
CL-USER> (DEFUN REV (L1)
(DO      ( (L2 NIL) )
( ( NULL L1) L2)
(SETQ L2 (CONS (CAR L1) L2) L1 (CDR L1) ) ) )
CL-USER> (rev '(1 2 3 4))
(4 3 2 1)
```

Замість значення локальної змінної можна задавати форму.

Приклад3. Ввести числовий список. Утворити новий список, елементи якого записано у зворотному порядку, а останнім елементом якого є довжина списку.

```
CL-USER> (Defun sum1 (L)
(DO      ( (L2 (LIST (Length L) ) ) )
( (NULL L) L2)
(SETQ L2 (CONS (CAR L) L2))
( SETQ L (CDR L) ) ) )
CL-USER>(Sum '(1 2 3 4))
(4 3 2 1 4)
```

Іноді треба зберегти значення S після виходу з циклу. Інакше S загубить своє значення. Додамо до означення функції, що знаходить суму елементів чистового списку вираз: (setq K S).

```
CL-USER> (DEFUN SUM (L)
(DO      ((S 0) )
( (NULL L) (setq K S))
(SETQ S (+ S (CAR L) ) L (CDR L) ) ) )
CL-USER>(Sum '(1 2 3 4))
10
CL-USER>K
10
```

Е.2 Спеціальна форма LOOP

Програма на мові Лісп має ієрархічну структуру з викликів функцій. Головна функція має виклики інших функцій. Кожна з функцій, що викликається у певний момент, також має виклики інших функцій. Всі поточні виклики зберігаються у стеку викликів.

Функція (*RETURN об'єкт*) повертає керування з поточної функції у функцію, яка її викликала, і передає їй результат роботи попередньої функції.

Для виходу з функції за певною умовою означення функції може поділятися на групи форм. Кожна група форм має мітку. Мітки встановлюють функцією (*BLOCK Мітка Форма1 Форма2 ...*). Блоки можуть вкладатися один у другий. По виконанню певної умови(наприклад збій), робота блоку з міткою переривається.

Деякі спеціальні форми самі організують блочну структуру означення функції. Наприклад, *COND*, *DO*. Для виходу з блоку застосовують функцію (*Return-from Мітка Об'єкт*). Функція (*Return-from NIL Об'єкт*) тотожна функції (*Return Об'єкт*). Для виходу за умовою функція *Return* або *Return-from* застосовується разом з спец формою *IF*. Вибір функції залежить від послідовності викликів у стеку.

Спеціальна форма *LOOP* обчислює форми у своєму тілі зліва направо. Дії повторюються до тих пір поки не буде виконана функція *RETURN* або *RETURN-FROM*, яка повертає результат виконання блоку. У одному означенні можливі різні виходи на наступний блок або з означення функції.

(*LOOP Форма1 ...ФормаN*), де серед форм знаходиться функція *RETURN*.

Приклад 5. Ввести числовий список. Одержати суму числового списку.

```
CL-USER>(DEFUN SUM (L) (SETQ S 0)
  (LOOP (IF (NULL L) (RETURN S))
    (SETQ S (+ S (CAR L) ))
    (SETQ L (CDR L) ) ) )
CL-USER> (SUM '(1 2 3 4))
```

Додаток Є

Робота з рядками та знаками

Є.1 Наслідування функцій

У Common Lisp тип рядка `STRING` як і тип `LIST` є підтипом більш загального типу `SEQUENCE`(послідовність).

На рисунку И.1 зображена ієрархія типів для типу `SEQUENCE`. Тип `STRING` можна розглядати, як одномірний масив і як тип вектор. Тип список є підтипом типу “послідовність” теж.

Більшість функцій типів, що знаходять у ієрархічній структурі вище, можна застосовувати до даних нижче розташованих типів.

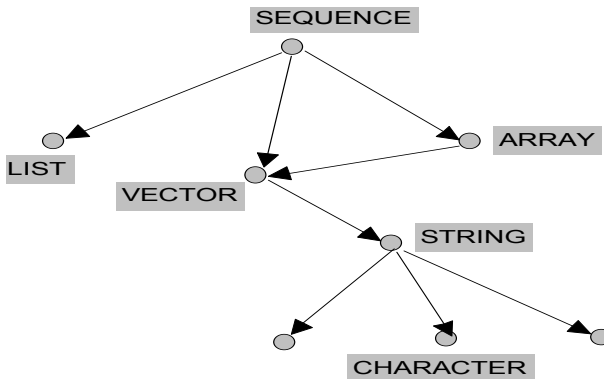


Рисунок К.1. Ієрархія типів для типу `SEQUENCE`

Наприклад, функції роботи з послідовностями `REVERSE` і `LENGTH` можна однаково застосовувати до списків і рядків:

```
CL-USER> (REVERSE '(q w e r))
```

```
(r e w q)
```

```
CL-USER> (REVERSE "qwer")
```

```
("rewq")
```

```
CL-USER> (LENGTH '(d f g h j))
```

```
5
```

```
CL-USER> (LENGTH "fghjk")
```

```
5
```

В той же час рядок складається із послідовності знаків як одномірний масив або вектор.

```
CL-USER> (count #\a "asadaf")
```

```
3
```

Завдання: Відсортувати рядок:

```
CL-USER> (sort "sawrb" 'string<)
```

```
"abrs w"
```

До функцій, що працюють з послідовностями відносяться: concatenate, count, count-if, find, find-if, subseq, position, position-if, reverse, sort, delete, delete-if, nsubstitute, nsubstitute-if.

Є.2 Типи знаку та рядку

Дане типу CHARACTER зображується з префіксом #. Наприклад, #\A, де префікс – показник на знак у середовищі.

Перевірити, чи є об'єкт типу CHARACTER можна за функцією предикатом:

```
CL-USER> (CHARACTERP #\g)
```

```
T
```

```
CL-USER> (CHARACTERP 'g)
```

```
NIL
```

```
CL-USER> (CHARACTERP '\g)
```

```
#\g
```

Перевірити чи є об'єкт рядком можна функцією STRINGP:

```
CL-USER> (STRINGP "asdf")
```

```
T
```

```
CL-USER> (STRINGP #\Q)
```

```
NIL
```

Є.3 Функції перетворення типів даних

Для перетворення типів даних застосовують наступні функції:

Функція *STRING* перетворює знак або символ у рядок:

```
CL-USER> (STRING #q)
```

```
"q"
```

```
CL-USER> (STRING 'q)
```

```
"Q"
```

Функція *INTERN* перетворює рядок у символ.

```
CL-USER> (INTERN "as")
```

```
|as| NIL
```

Функція *COERCE* використовується для підтипів типу "послідовність" перетворює об'єкт, що стоїть першим аргументом до типу, що вказаний другим аргументом.

```
CL-USER> (COERCE "qwe" 'LIST)
```

```
(#\q #\w #\e)
```

```
CL-USER> (COERCE '(\Q #\W #\E) 'STRING)
```

```
"QWE"
```

```
CL-USER> (COERCE „A" 'CHARACTER)
```

```
#\A
```

Функцією *COERCE* не можна перетворити знак у рядок.

Функція перетворює код у знак:

```
CL-USER> (CODE-CHAR 97)
```

```
#\a
```

Функція перетворює знак у код:

```
(CL-USER>CHAR-CODE #\a)
```

```
97
```

Є.4 Дії над рядками

Є.4.1 Функції порівняння рядків та символів

Функції групи `STRING=`, `STRING<`, `STRING>`, `STRING<=`, `STRING>=` виконують лексиграфічне порівняння рядків або символів (порівняння за кодами). Якщо рядки не однакової довжини, то менший рядок доповнюється кодом 0.

Функція `STRING=` порівнює рядки або символи і повертає `T` або `NIL`. Функція враховує регістр.

```
CL-USER> (STRING= "a" "A")
```

```
NIL
```

```
CL-USER> (STRING= 'a 'A)
```

```
T
```

Функція `STRING-EQUAL` дозволяє ігнорувати регістр.

```
CL-USER> (STRING-EQUAL "a" "A")
```

```
T
```

Інші функції групи перевіряють тест у назві функції і враховують регістр. Якщо аргументи не задовольняють тесту, то повертають `NIL`, інакше повертають позицію у рядку починаючи з якої тест виконується.

```
CL-USER> (STRING< "A" "a")
```

```
0
```

```
CL-USER> (STRING< 'ca 'CB)
```

```
1
```

```
CL-USER> (STRING> 'CA 'CA)
```

```
NIL
```

```
CL-USER> (STRING<= 'AS 'BS)
```

```
0
```

```
CL-USER> (STRING>= 'AS 'AS)
```

```
2
```

Функція `STRING LESSP` ігнорує регістр:

```
CL-USER> (STRING-LESSP "a" "A")
```

NIL

Є.4.2 Функції відбору

Функція *CHAR* повертає знак з рядка за позицію. Рядок трактується як одномірний масив. Знаки рядка нумеруються з 0. Якщо покажчик виходить за межі рядка, то функція повертає NIL

```
CL-USER> (CHAR "qwe" 2)
```

```
#\e
```

Функція *SUBSEQ* одержує з рядку підрядок за вказаними позиціями. Якщо позиція одна, то підрядок вибирається від позиції до кінця рядку. Якщо перша позиція більше довжини рядка, повертається порожній рядок. Функція не змінює сам рядок.

```
CL-USER> (SUBSEQ "qwert" 2 3)
```

```
"e"
```

```
CL-USER> (SUBSEQ "qwert" 2 )
```

```
"ert"
```

```
CL-USER> (SUBSEQ "qwert" 5)
```

```
""
```

Є.4.3 Заміна знаку рядка

Замінити знак у рядку можна за функцією привласнювання SETF.

```
CL-USER> (SETQ N "qwe" )
```

```
CL-USER> (SETF (CHAR N 2) #\k)
```

```
#\k
```

```
CL-USER>N
```

```
"qwk"
```

У прикладі функція (CHAR N 2) повертає #\e. Тоді вираз 2 запишеться: (SETF #\e #\k) => #\k. Вираз замінює знак #\e на знак #\k. Звертаючись до змінної N, одержимо змінений рядок "qwk".

Є.4.4 Функція створення рядку

Утворити рядок повторюванням однакових символів можна за функцією MAKE-STRING. Функція повертає рядок.

```
CL-USER> (MAKE-STRING 5 :INITIAL-ELEMENT #\7)
"77777"
```

Є.4.5 Функції зміни регістру

Змінити регістр знаків рядку можна функціями STRING-UPCASE і STRING-DOWNCASE. Функції мають опції, що вказують з якої і по яку позицію треба змінити регістр. Функції повертають змінений рядок.

```
CL-USER> (STRING-UPCASE "asdef" :start 1 :end 3)
"aSDef"
CL-USER> (STRING-DOWNCASE "ASDFG" :start 1 :end 3)
"AsdFG"
```

Є.4.6 Функція усікання рядку

Функція відсікає з рядка від початку і від кінця вказаний підрядок. Функція повертає змінений рядок.

```
CL-USER> (STRING-TRIM "AS" "ASDAS")
"D"
```

Є.4.7 Функція пошуку

Функція шукає підрядок у рядку і повертає позицію початку першого знайденого підрядку. Якщо підрядку немає повертає NIL.

```
CL-USER> (SEARCH "qw" "erqwtqw")
2
```

Є.4.8 Функція об'єднання рядків

Функція дозволяє об'єднати довільну кількість послідовностей одного типу.

Приклад 1. Об'єднати три рядки.

```
CL-USER> (Concatenate 'string "as" "df" "gh")
```

“asdfgh”

Приклад 2. Об'єднати два списки.

CL-USER> (Concatenate 'list '(1 2) '(3 4) '(5 6))
(1 2 3 4 5 6)

Є.5 Робота з властивостями символів

Записати значення властивості символу можна за функціями:

CL-USER> (SETF (GET 'книга 'автор) 'Саймак)

Саймак

Одержати значення за властивістю символу можна функцією:

CL-USER> (GET 'книга 'автор)

Саймак

Одержати список властивостей символу можна за функціями:

(CL-USER> (SYMBOL-PLIST 'книга)

(автор Саймак)

Додаток Ж

Функції вводу-виводу

Ж.1 Функції вводу

Функції вводу реалізують увід з чеканням. Результат функції розміщують на місце виклику. Функції подаються:

(Функція „Вхідний_потік” Дія_за_помилкою), де “Вхідний потік” ім’я потоку, який призначається на файл або пристрій; Дія за помилкою вказує, яку треба виконати дію при вводі з помилкою. Всі аргументи можуть братися за замовчанням. Замовчення для потоку екран.

Функція *Read* є універсальною функцією вводу даних будь-якого типу. Функція вводить S - вираз по натисненню „Enter”. Якщо парної дужки нема, то увід дужки чекається.. Функція перетворює дане до типу за синтаксисом.

Наприклад:(CL-USER> (+ (read) 2)

4

6

Функція *Read-line* вводить рядок до „Enter” і повертає рядок.

Наприклад:(CL-USER> (setq A (read-line))

4

“4”

Функція *Read-Char* вводить знак.

Наприклад:(CL-USER> (setq A (read-char))

4

#\4

Функція *Read-byte* вводить байт з бінарного файлу.

(read-byte input)

Ж.2 Функції виводу

Функції виводу реалізують вивід об’єкту. Функції подаються:

(Функція Об’єкт „Вихідний_потік” Дія_за_помилкою), де “Вихідний потік” ім’я потоку, який призначається на файл або

пристрій; Дія за помилкою вказує, яку треба виконати дію при виводі з помилкою. Аргументи можуть братися за замовчанням крім першого. Замовчення для потоку клавіатура.

Функція PRIN1 базова універсальна виводить об'єкти будь-якого типу. Після виводу курсор залишається у наступній позиції.

Наприклад: CL-USER> (prin1 23)

23

23

Функція PRINT модифікація базової PRIN1 виводить об'єкти будь-якого типу. Перед виводом курсор переводиться у наступний рядок. Після виводу об'єкту виводиться пропуск.

Наприклад: CL-USER> (print "qw")

"qw"

"qw"

Функція PRINC модифікація базової PRIN1 виводить об'єкти будь-якого типу. Працює як PRIN1, але об'єкт виводить без парних розділових знаків.

Наприклад: CL-USER> (princ "qw")

qw

"qw"

Функція WRITE-LINE виводить рядок, інакше помилка. Вивід рядку без розділових знаків. Після виводу курсор переводиться у наступний рядок.

Наприклад: CL-USER> (Progn (write-line "qwe") (write-line "asd"))

qwe

asd

"asd"

Функція WRITE-STRING виводить рядок , інакше помилка. Вивід рядку без розділових знаків. Після виводу курсор переводиться у наступну позицію.

Наприклад: CL-USER> (Progn (write-string "qwe") (write-string "asd"))

qweasd

"asd"

Функція WRITE-CHAR виводить знак, інакше помилка. Вивід рядку без префіксу. Після виводу курсор переводиться у наступну позицію.

Наприклад: CL-USER> (write-char #\2)

2

#\2

Функція WRITE-BYTE виводить байт, інакше помилка. Після виводу курсор переводиться у наступну позицію.

(write-byte 65 output)

Наприклад: (CL-USER> (setq A (read-byte in NIL))

Додаток 3

Робота з файлами

3.1 Функції роботи з файлами

Мова Common Lisp дозволяє працювати з файлами послідовним та прямим методом доступу.

Файли можуть бути символьними та бінарними. Для роботи з файлами та пристроями застосовується поточна організація.

Функція OPEN: (OPEN “Ім’я файлу” параметри) застосовується для відкриття файлу. Вона повертає потік для роботи з даними файлу.

Файл може відкриватися для читання або запису, читання – запису. Для вказівки типу роботи з файлом існує параметр :Direction, значення якого вказує на тип роботи.

Значення означає: INPUT – читання; :OUTPUT – запис; :IO – читання-запис.

Якщо застосувати тільки значення параметру Direction, то можлива помилка.

При читанні файла може застосовуватись параметр :IF-DOES-NOT-EXIST. Значеннями цього параметру можуть бути :ERROR замовчення; :CREATE утворити файл з вказаним ім’ям і відкрити його; NIL, якщо файл відсутній ввести з клавіатури.

При читанні бінарного файлу застосовують параметр :element-type із значенням ‘(unsigned-byte 8).

При запису даних у файл функція OPEN має параметр :IF-EXISTS, значення якого вказують :SUPERSEDE можливо замінити існуючий файл; :APPEND доповнювати файл; :OVERWRITE перезапис файлу; NIL повертається NIL замість потоку.

При запису бінарного файлу застосовують параметр :element-type із значенням ‘(unsigned-byte 8).

Наприклад, відкриємо файл для читання символьного файлу:

```
(SETQ N (OPEN "A.TXT" :DIRECTION :INPUT :IF-DOES-NOT-EXIST :CREATE))
```

Перевірити наявність у потоці даних можна функцією:

(LISTEN Потік). Функція повертає Т, якщо у потоці є знаки інакше NIL.

Закрити файл можна функцією: (CLOSE потік)

Функція (FILE-POSITION потік № байту) працює у двох режимах:

Якщо вказано тільки потік, то повертає поточний покажчик на байт).

Якщо вказано потік і № байту , встановлює покажчик на байт.

Функція (FILE-LENGTH потік) визначає розмір файлу у байтах.

3.2 Приклади роботи з файлами

3.2.1 Читання даних з файлу

1. Програма вводить вміст файлу „a.txt” і виводить його на екран, якщо файлу немає, то виводить на екран повідомлення.

```
CL-USER> (defun s ()(setq N (OPEN "r.txt" :DIRECTION :INPUT
:IF-DOES-NOT-EXIST NIL) )
```

```
(Loop(if (EQ N NIL)(Progn (Print "NO File")(return-from s)))
```

```
(If (Not (Listen N)) (Progn (Close N)(Return-from s)))
```

```
(setq a1 (READ N))(princ a1) ))
```

2. Програма читає дані з файлу і виводить їх на екран, якщо файлу немає то утворюється порожній файл.

```
CL-USER> (defun s () (setq N (OPEN "r.txt" :DIRECTION
:INPUT :IF-DOES-NOT-EXIST :create ) )
```

```
(Loop (IF (Not(Listen N))(Progn (Close N) (return-from s )))
```

```
(setq a1 (READ N))(princ a1)))
```

3. Програма читає бінарний файл і виводить його вміст на екран. Файл утворений блокнотом.

```
CL-USER> (defun s () (setq N (OPEN "r.txt" :DIRECTION :Input
:element-type '(unsigned-byte 8) ) )
  (Loop (IF (Eq (File-position N)(file-length N))(Progn (Close N)
(return-from s )))
    (setq a1 (Read-byte N))(Print a1)))
```

3.2.2 Запис даних у файл

1. Програма створює файл. Якщо файл існує, то нічого не робить.

```
CL-USER> (defun s () (setq N (OPEN "a.txt" :DIRECTION
:Output :IF-EXISTS NIL) )
  (IF (NULL N)(return-from s))
  (Loop (setq a1 (Read-line) )
    (If (Equalp a1 "0" )(Progn(Close N)(return-from s)))
    (princ a1 N)(TERPRI N)) )
```

2. Програма перезаписує файл, якщо він існує.

```
CL-USER> (defun s () (setq N (OPEN "a.txt" :DIRECTION :Output
:IF-EXISTS :overwrite) )
  (setq a1 (READ))(setq a2 (read ))(setq a3(read ))(Print a1 N)(Print a2
N)(Print a3 N)
  (CLOSE N))
```

3. Програма дописує файл, якщо файл існує.

```
CL-USER> (defun s () (setq N (OPEN "ka.TXT" :DIRECTION
:Output :IF-EXISTS :append) )
  (setq D1 (+ 1 (file-length N))) (file-position N D1)
  (setq a1 (READ))(setq a2 (read ))(setq a3(read ))
  (Print a1 N)(Print a2 N)(Print a3 N)
```

(CLOSE N))

4. Програма створення бінарного файлу. Програма читає знаки з клавіатури функцією Read-char і перетворює їх у коди. Коди записуються у файл функцією Write-byte. Керуючі символи (VK) і ПС також записуються у файл.

Функція Print видає на екран символи за кодами.

```
CL-USER> (defun s () (setq N (OPEN "r.txt" :DIRECTION :Output
:element-type '(unsigned-byte 8)) )
```

```
(setq a1 (char-code(READ-char)))
```

```
(setq a2 (char-code(read-char )))
```

```
(setq a3(char-code(read-char) ))
```

```
(Print (write-byte a1 N))
```

```
(Print(write-byte a2 N))
```

```
(Print(write-byte a3 N)) (CLOSE N))
```

Результат роботи програми залежить від наявності файлу з вказаним ім'ям. Крім того, після помилки треба закривати вручну файл.

Функції можуть виконувати дії не тільки над даними файлу, а над самими файлами.

Функція PROBE-FILE перевіряє, чи існує файл у вказаному каталозі:

```
CL-USER> (PROBE-FILE "D:\\A.txt")
```

```
#P"D:\\A.txt"
```

Якщо файл не існує, то повертається NIL, інакше повертається шлях до файлу.

Функція DELETE-FILE вилучає файл з вказаного каталогу.

```
CL-USER> (DELETE-FILE "D:\\A.txt")
```

```
#P"D:\\A.txt"
```

При роботі з файлами треба добре розуміти, що запис на диск або читання з диску виконується через буфер. Буфер автоматично записується на диск тільки після його заповнення. Тому для запису залишку даних на диск застосовується функція CLOSE або ці дії виконує функція WITH –OPEN-FILE.

Важливо також знати, дані передаються через потоки, тому після читання або запису у потоці може залишатися сміття. Для наступного використання того ж самого потоку треба очищувати потік.

Вхідний потік очищується функцією:

CL-USER> (CLEAR-INPUT Потік).

Вихідний потік очищується функцією:

CL-USER> (CLEAR-OUTPUT Потік).