# Лабораторна робота 1. Обчислення виразів

Deadline вересень

## Теоретичні відомості

### Характеристика функціональних мов програмування

Основним поняттям функціонального програмування є функція. Функціональна програма є набором визначень функцій. Функції визначаються через інші функції або рекурсивно через самих себе. У процесі виконання програми функції отримують параметри, обчислюють і повертають результат, у випадку необхідності обчислюються значення інших функцій. Програмуючи функціональною мовою, програміст не повинен описувати порядок обчислень. Йому необхідно просто описати бажаний результат у вигляді системи функцій.

Строго функціональне програмування не має операцій присвоєнь та засобів передачі керування. Повторні обчислення здійснюються за допомогою рекурсії, яка є основним засобом функціонального програмування.

Серед важливих властивостей функціонального програмування є такі:

1. Представлення програми і даних відбувається через списки. Це дає змогу програмі обробляти інші програми і навіть саму себе.
2. Функціональні мови, як правило, є інтерпретуючими мовами.
3. Функціональні мови є безтиповими, це означає, що символи не зв’язуються за замовчуванням з яким-небудь типом.
4. Функціональні мови мають незвичний синтаксис через велику кількість дужок.
5. Функціональні програми, написані для обробки символьних даних, є набагато коротші, аніж написані імперативними мовами/

### Функції-примітиви мови Scheme

Вважається, що строго функціональна мова програмування має дуже обмежене число базових функцій, на основі яких можна побудувати всі інші функції. Як правило, вибирається сім примітивних функцій. Ця обмеженість є важливою в теоретичному плані в галузі програмування, оскільки вирішується проблема розв’язання довільної задачі з допомогою обмеженого набору примітивних функцій.

Виклик функцiї у функціональних мовах програмування здійснюється у формі списку і має такий формат:

(function\_name arg1 arg2 ... argN),

Де function\_name – iм'я функцiї,

arg1,arg2,...argN – її аргументи.

Базові функції функціональної мови програмування Scheme наведено в табл. 1.

Таблиця 1. Базові функції функціональних мов програмування

|  |  |
| --- | --- |
| **Мова Scheme** | **Семантика** |
| QUOTE | Функція використовується для блокування обчислень, що призводить до інтерпретації частини S-виразу не як програми, а як даних |
| **(CAR object)** | Вибирає голову непорожнього списку, пари або неправильно-сформованого списку |
| **(CDR object)** | Вибирає хвiст непорожнього списку, пари або неправильно-сформованого спику |
| **(CONS object1 object2)** | Функція-конструктор - об'єднює об'єкти 1 та 2 |
| **(EQ? atom1 atom2)** | Порiвнює два об’єкта на тотожність |
| **LIST? object)** | Перевіряє чи є object списком. |
| (**PAIR? object)** | Перевіряє чи є object парою |
| **(СOND (condition1 action1) (condition2 action2)… (conditionN actionN))** | Набуває значення з множини значень action1, action2, ..., actionN в залежності від значень умовних виразів condition1, condition2, …, conditionN. |

Таблиця 2. Деякі математичні функції

|  |  |
| --- | --- |
| **Мова Scheme** | **Семантика** |
| **ABS** | функція знаходить модуль числа. |
| **EXP** | функція, яка повертає ex. |
| **LN** | функція, яка повертає натуральний логарифм аргумента. |
| **SIN** | функція, яка повертає синус аргумента. |
| **COS** | функція, яка повертає косинус аргумента. |
| **SQRT** | функція, яка повертає квадратний корінь аргумента. |

Таблиця 3. Предикати

|  |  |
| --- | --- |
| **NUMBER?** | предикат, який перевіряє чи аргумент є числом |
| **POSITIVE?** | предикат, який перевіряє чи аргумент є додатнім числом. |
| **NEGATIVE?** | предикат, який перевіряє чи аргумент є від’ємним числом |
| **EVEN?** | предикат, який перевіряє чи число є парним. |
| **ODD?** | предикат, який перевіряє, чи число є непарним. |
| **NULL?** | предикат, який перевіряє чи аргумент є порожнім списком. |

## Середовище функціонального програмування Dr.Racket

Оскільки в середовищі Dr.Racket підтримується ряд діалектів та стандартів, оберемо класичний стандарт Scheme R5RS. Для цього потрібно обрати меню Мова-> Обрати мову ->Other languages-> R5RS.

На рис. 1 наведено вигляд головного вікна середовища.

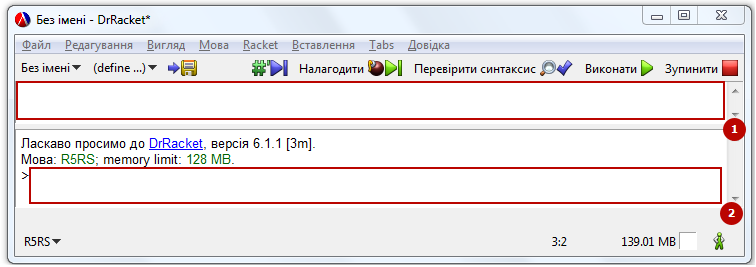


Рис. 1. Головне вікно середовища.

Головне вікно середовища розробки поділене на дві частини: 1 – область для вводу тексту програми; 2 – область командного рядка. В першій частині зручно писати текст програми і всі виклики, в другій – виклики функцій.

На панелі інструментів винесено кнопки для від лагодження, перевірки синтаксису, запуску на виконання та зупинки програми (рис. 2).



Рис. 2. Панель інструментів в середовищі Dr.Racket

Приклад 1. Обчислимо суму 1 + 2. На мові Scheme потрібно написати (+ 1 2). Після цього запустимо програму на виконання клавішою F5 або кнопкою «Запустити». Результат виконання програми відобразиться в другій половині екрану (рис. 3).

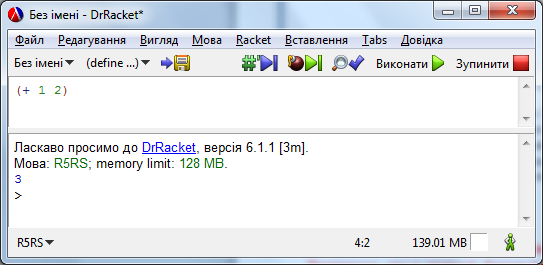


Рис. 3. Виконання програми в середовищі Dr. Racket

Є можливість виконувати команди безпосередньо в командному рядку: після введеного тексту програми потрібно натиснути клавішу Enter. Перемножимо числа 2, 3, 4: для цього введемо в командний рядок текст (\* 2 3 4) і натиснемо клавішу Enter. Наступний рядок міститиме шуканий результат (рис. 4).

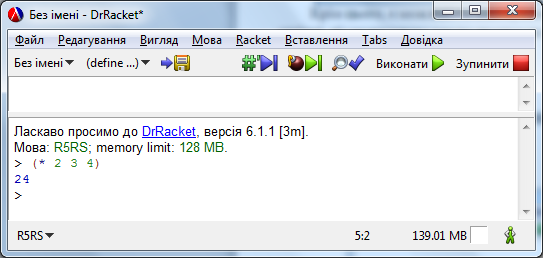


Рис. 4. Виконання програми в командному рядку

Щоб поставити коментар потрібно перед текстом поставити символ «;». Scheme не є чутливим до регістру і до кількості пробілів між об’єктами. Вирази (a) та ( a ) є тотожними.

На рис. 5 наведено приклад використання змінних та коментарів.

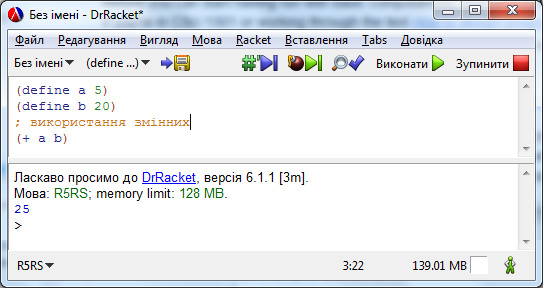


Рис. 5. Використання змінних та коментарів у середовищі Dr. Racket

### Процес налаштування коду в середовищі Dr.Racket

Приклад 2. Обчислимо вираз: **(+ 2 (\* 3 4))**. Замість кнопки «Виконати» натиснемо кнопку «Налагодити». Після цього головне вікно буде мати вигляд як на рис. 6.

Вікно має такі основні частини: 1 – текст програми, 2 – поточний результат виконання програми, 3 – меню для налаштування, 4 – вікно із відображенням вмісту стеку, 5 – вікно для відображення поточних значень змінних.

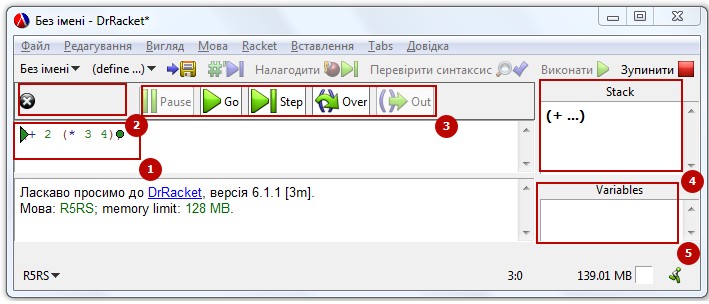


Рис. 6. Інтерфейс середовища Dr.Racket у режимі відлагодження

Спочатку в стеку є наявна функція додавання (рис. 14). Курсор відлагодження вказує на цю функцію. Натиснемо кнопку «Step». В стеку з’явилась функція множення (рис. 7). Курсор налаштування вказує на цю функцію у вікні тексту програми.

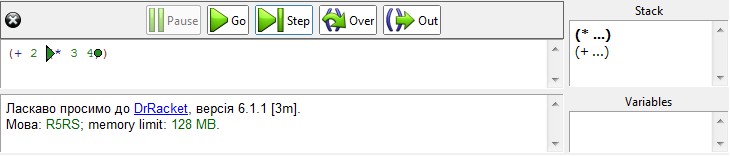


Рис. 7. Додавання в стек функції множення

Натиснемо ще раз кнопку «Step». Оскільки далі за функцією множення немає інших функцій, відбудеться обчислення множення за вказаними параметрами 3 і 4. Проміжний результат (число 12) відображено у відповідній частині екрану (рис. 16).

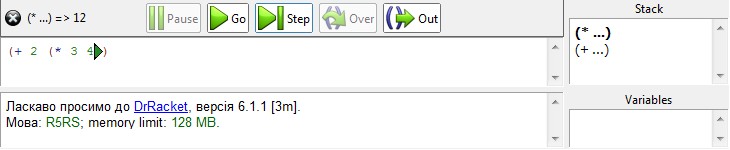


Рис. 8. Здійснення обрахунків для функції множення

Натиснемо ще раз кнопку «Step». Як видно із стеку, функція множення виконалась і вийшла з стеку. Курсор налаштування вказує на кінець виклику функції додавання. Результат виконання функцій (число 14) відображено в області біля меню налаштування (рис. 9).

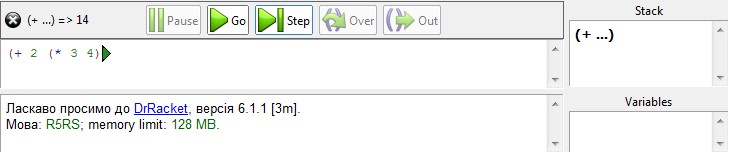


Рис. 9. Здійснення обрахунків для функції додавання

Чергове натискання кнопки «Step» завершує виконання програми – з стеку вивільняється функція додавання, а остаточний результат відображається у вікні командного рядка (рис. 18).



Рис. 18. Завершення виконання відлагодження програми

## **Завдання**

Обчислити задані вирази відповідно до варіантів при заданих користувачем значеннях параметрів A, B,C,D.

|  |  |
| --- | --- |
| 1 X=(2A - B2)/(D(SinC - 4.5)) | 15. X=((1 - A)CeA)/((1-B)CosD) |
| 2 X=(2.3 + AB2)/(0.4 + CeD) | 16. X=(ACosA(1 + B3))/(C(1 - D2)) |
| 3. X=(-2.25(A + 2BC))/(B - D1/2) | 17. X=A/(1 + B/(C + Ln(D + A))) |
| 4. X=2ASin(B/2) + DC1/2 | 18. X=(CosA + SinB)/(LnC + LnD)1/2 |
| 5. X=(A - |B/C|)/(2.75Sin2D) | 19. X=(ACos2B + 1)1/2LnC/D |
| 6 X=(|A - B|2C2)/(1 - D2) | 20. X=Arctg(4A/(B2C) - 1)1/2 |
| 7. X=e((SinA + SinB)/(C + D)) | 21. X=A2 + |SinB/(C - D)1/2| |
| 8. X=(1 - A)/(1 + A)+|(B - 2D)/C2| | 22. X=(|A| + |B|)2/(D - C1/2) |
| 9. X=(A2-B2)(C-D)/(e(A2 - B2))1/2 | 23. X=(A1/2 + B2)eC+D/(|A| - |B|) |
| 10. X=(1-B)/(1+B)+ |(C-2A)/C2| | 24. X=1/2Ln((1+SinA)/(1- SinB)2) |
| 11. X=Sin3ACos2B/(eC + 1 + SinD) | 25. X=Sin3CCos2A/(eA + 1 + SinB) |
| 12. X=|1-B||1-A|/((1+SinC)(1+CosC))1/2 | 26. X=(1/(A2-B2)1/2)+(3C3/(C2-D2)1/2) |
| 13. X=(ASinB+BCosA)/(1-SinC\*|B+D|) | 27. X=(eACos(B/2+D/C))/(DC+SinC) |
| 14. X=A2/|1-B|+B2/|1-D|+C2/LnA | 28. X=(A-4B)2/(1+(1+SinC)(1+CosC)) |

# Лабораторна робота 2. Використання рекурсії для організації повторювальних процесі

Deadline вересень

## Теоретичні відомості

### Визначення процедур

Для створення функції з іменем потрібно використати один із форматів опису функції:

(define <name> (lambda (<arguments>) (<body>))),

або

(define (<name> <arguments>) (<body>)),

де **<name>** - ім’я функції; **<arguments>** - список аргументів (через пробіл); **<body>** - правильний S-вираз, який набуває значення.

*Приклад*.

(define (f x)

(+ x 42))

Виклик функції: (f 23)

Результат 65

Процедура, в попередньому абзаці, є абстракцією виразу за допомогою об'єктів. У першому визначенні прикладу визначена процедура, названа f. (Зверніть увагу на круглі дужки навколо f x, що позначають, що це -визначення процедури.) Вираз (f 23) є викликом процедури, приблизно означає "вирахувати (+ х 42) (тіло процедури) з x, прив'язаним до 23 ".

Оскільки процедури є об'єктами, їх можна передавати в інші процедури:

(define (f x)

(+ x 42))

(define (g p x)

(p x))

Виклик : (g f 23)

Результат: 65

У цьому прикладі тіло g обчислюється з p, прив'язаним до f, і x, що прив'язаний до 23, що еквівалентно (f 23) і обчислюється в 65.

Фактично багато операцій Scheme забезпечуються не синтаксисом, а змінними, значеннями яких є процедури. Операція +, наприклад, в Scheme є всього лише регулярним ідентифікатором, пов'язаним з процедурою, що додає числові об'єкти. Те саме стосується і \*, і багатьох інших:

(define (h op x y)

(op x y))

(h + 23 42) =) 65

(h \* 23 42) =) 966

### Хвостова рекурсія

Це випадок рекурсії, коли рекурсивний виклик функції відбувається наприкінці її роботи. Використовується у мовах програмування для оптимізації, через можливість заміни виклику функції на ітерацію, без використання стеку. Ця оптимізація широко використовується у функціональних мовах програмування.

Коли відбувається виклик функції комп'ютер має запам'ятати адресу повернення, щоб після завершення викликаної функції повернутися і продовжити виконання програми. Зазвичай адреса виконання зберігається у стеку. Іноді, остання дія функції після завершення всіх інших операцій, це просто виклик функції, можливо самої себе, і повернення результату. В цьому випадку немає необхідності запам'ятовувати адресу повернення, нововикликана функція буде повертати результат безпосередньо за адресою повернення записаною для початкової функції.

Приклад на [Scheme](https://uk.wikipedia.org/wiki/Scheme):

(**define** (factorial n)

(**define** (fac-times n acc)

(**if** (= n 0)

acc

(fac-times (- n 1) (\* acc n))))

(**if** (< n 0)

(display "Невірний параметр!")

(fac-times n 1)))

\

Звичайний рекурсивний спосіб обчислення факторіала, наведений нижче, **не** є хвостовий-рекурсивним, оскільки в кожному виклику функції після рекурсивного виклику виробляються додаткові операції, а саме множення на *n.*

Приклад на [Scheme](https://uk.wikipedia.org/wiki/Scheme):

( define ( factorial n )

( if ( = n 0 )

1

( \* n ( factorial ( - n 1 ) )

) ) )

## Завдання

**Написати процедури, що обчислюють задану функцію за допомогою рекурсивного процесу. Продемонструвати застосування звичайної та хвостової рекурсії.**

**Варіант 1.** Знайти суму цифр натурального числа *n* та значення глибини рекурсії, використовуючи рекурентне означення функції *f(n)*:



*Підказка*. Умова продовження рекурсії: сума цифр числа дорівнює останній цифрі плюс сума цифр числа без останньої цифри (числа, що ділиться без остачі на 10). Умова закінчення рекурсії: якщо число дорівнює 0, то сума його цифр дорівнює 0.

Варіант 2. Знайти кількість одиниць в двійковому представленні числа *n* та значення глибини рекурсії*,* використовуючи рекурентне означення функції *f(n)* (& - операція побітового логічного множення):



**Варіант 3.** Знайти значення біноміального коефіцієнта при заданих *n, k* за формулою



використовуючи рекурентне співвідношення



**Варіант 4.** Знайти значення функції Аккермана *A(m, n),* використовуючи рекурентне співвідношення:



**Варіант 5.** Для заданого числа визначити рекурсивні процедури для обчислення суми та кількості його цифр, максимальної та мінімальної цифри. Визначити рекурентні співвідношення та глибину рекурсії.

**Варіант 6**. Функція f визначається правилом:



Напишіть процедуру, яка обчислює f за допомогою рекурсивного та ітеративного процесів.

**Варіант 7.** Наведена нижче таблиця називається трикутником Паскаля .

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

. . .

Всі числа по краях трикутника рівні 1, а кожне число всередині трикутника дорівнює сумі двох чисел над ним. Напишіть процедуру, яка обчислює елементи трикутника Паскаля за допомогою рекурсивного процесу.

**Варіант 8.** Синус кута (заданого в радіанах) можна обчислити, якщо скористатися наближенням ***sin x ≈x*** при малих ***x*** і вжити тригонометричну тотожність для зменшення значення аргументу ***sin***. (кут «достатньо малий », якщо він не більше 0.1 радіана.



**Варіант 9**. Розробіть процедуру, яка породжує ітеративний процес для множення двох чисел за допомогою додавання, подвоєння і ділення навпіл, і витрачає логарифмічну кількість кроків. Цей алгоритм називають «методом російського селянина»

**Варіант 10**. Потрібно сплатити поштове відправлення, вартість котрого складає ***m*** копійок, а в наявності тільки поштові марки номіналом ***x,y,z*** копійок. Скількома різними способами можна сплатити поштове відправлення? Розробити рекурсивну процедуру для обчислення кількості зображень числа ***m*** у вигляді суми певних фіксованих чисел з використанням рекурентних співвідношень.

*Підказка*. Використати рекурентне співвідношення для чисел Фібоначчі.

**Варіант 11**. Реалізувати генератор послідовності псевдовипадкових чисел {*Vi*} на основі рекурентного співвідношення *Vi* = (*aVi*–1 + *bVi–*2 + *c*) mod *m*, де *a*, *b*, *c*, *m* — довільні натуральні параметри. Перші два значення, *V*1 і *V*2,задаються випадково. Підібрати значення параметрів, за яких послідовність є схожою на випадкову.

**Варіант 12.** Написати рекурсивну процедуру, яка методом ділення відрізка навпіл (методом дихотомії) знаходить з точністю *eps* корінь *рівняння* ***f(x)=sinx – cosx=*0** на відрізку [*a,b*] (eps>0, a<b, f(a)\*f(b)<0). Згідно з методом дихотомії, якщо *f(a)* i *f(b)* мають різні знаки, то між *точками a,b* існує корінь *x*. Нехай *m* – середня точка в інтервалі [*a,b*]. Якщо *f*(*m*)=0, то корінь *x=m*. Якщо *f*(*m*)<>0, то або *f*(*a*) і *f*(*m*) мають різні знаки, або *f*(*m*) і *f*(*b*) мають різні знаки. Якщо *f*(*a*)\**f*(*m*)<0, то корінь лежить в інтервалі *a≤x≤m* Інакше він лежить в інтервалі m≤x≤b Процес продовжується доти, поки інтервал не стане менший eps.

**Варіант 13**. Визначити рекурсивну функцію обчислення степеня дійсного числа з цілим показником xn*,* згідно з рекурентним співвідношенням:



**Варіант14**. Якщо ***n*** - просте число, та ***a*** - довільне ціле число менше, ніж ***n***, то ***a***, зведена в степінь ***n***, дорівнює ***a*** за модулем ***n***. Написати процедуру за алгоритмом перевірки числа на простоту: маючи число ***n***, випадковим чином вибрати число ***a <n*** і обчислити залишок від ***аn***по модулю ***n***. Якщо цей залишок не дорівнює ***a***, то ***n*** безумовно не є простим. Якщо він дорівнює ***a***, то **n** просте. Тоді потрібно взяти ще одне випадкове a і перевірити його тим же способом. Якщо і воно задовольняє рівняння, то n просте.

**Варіант 15.** По колу стоять n людей, яким присвоєні номери від 1 до n. Починаючи відлік з першого і рухаючись по колу, кожна друга людина виходитиме з кола доти, поки не залишиться одна. Нехай номер того, хто залишився, x. Потім по колу стоятиме x людей і процедура виходу з колу людей повторюватиметься доти, поки не залишиться одна людина з номером y. Ці процедури повторюватимуться доти, поки номер тої людини, що залишиться, не стане рівним первинній кількості людей в потоковому раунді. Визначити кількість повторень процедури виходу людей з кола після першої ітерації та номер людини, яка залишилася.

*Підказка*. Використати рекурсію, визначивши рекурентне співвідношення/

**Варіант 16.** По заданим *b, p, m* обчислити значення виразe I:\!TeachingStudentKPI2018\course1_A&P\A&Psemestr1\LabA&P_semestr1\MethodLabWorkC++Semestr1Html\labs\lab_05\pictures\var23_1.gifmod m. Обчислення I:\!TeachingStudentKPI2018\course1_A&P\A&Psemestr1\LabA&P_semestr1\MethodLabWorkC++Semestr1Html\labs\lab_05\pictures\var23_1.gif виконати, використовуючи алгоритм, що базується на двійковому розкладанні показника степені p:



**Варіант 17.** Визначена рекурсивна функція f(n) через рекурентне співвідношення:

****

**Варіант 18.** Визначити рекурсивні функції зображення натурального числа у двійковій, вісімковій системах числення та у системі числення з основою N > 1 та глибину рекурсії. Числа у десятковій системі числення вводити з клавіатури.

**Варіант 19.** Побудувати 2 варіанти функції визначення чисел Фібоначчі, використовуючи звичайну рекурсію та “хвостову”.

**Варіант 20.** Побудувати 2 варіанти функції обчислення факторіалу, використовуючи звичайну рекурсію та “хвостову**”.**

# Лабораторна робота 3. Форми lambda та let

Deadline жовтень

## Теоретичні відомості

Для визначення нової функції мовою Scheme використовується така конструкція:

(lambda (<arguments>) (<body> )),

де **<arguments>** - список аргументів (через пробіл); **<body>** - правильний S-вираз, який набуває значення.

Такий запис функцій називається лямбда-функцією, оскільки функція немає імені для її зручного виклику. Наприклад, визначемо функцію одного аргументу, яка додає до переданого аргументу число 10:

( lambda ( x ) ( + x 10 ) )

Згідно концепції Lisp, всі функції викликаються у форматі

**(function\_name param1 param2 … paramN)**

Якщо функція є без імені, то можна підставити замість **function\_name** сам опис функції. Наприклад, виклик вищеописаної функції для параметру 20:

( ( lambda ( x )( + x 10 ) ) 20 )

Недоліком такого опису функції є складність її виклику, оскільки для цього потрібно завжди використовувати повний опис функції.

Для створення функції з іменем потрібно використати один із форматів опису функції:

### (define <name> (lambda (<arguments>) (<body>)))

де **<name>** - ім’я функції; **<arguments>** - список аргументів (через пробіл); **<body>** - правильний S-вираз, який набуває значення.

Приклад.

((lambda(x)

(+ x 42))

23)

Правилами для ключового слова lambda визначено, що перша підформа є списком параметрів, а решта підформи - тілом процедури.

Змінні Scheme, прив'язані визначеннями або виразами **let** або **lambda**, фактично прив'язуються не безпосереднє до об'єктів, що визначені у відповідних прив’язках, а до адреси пам'яті, що містить ці об'єкти. Вміст за цими адресами згодом може бути змінено за допомогою присвоювання:

(let ((x 23))

(set! x 42)

x)

У даному випадку тіло виразу let складається з двох обчислюваних послідовно виразів зі значенням остаточного виразу, що приймає значення всього виразу let. Вираз (set! х 42) є присвоєнням, що вказує "замінити об'єкт за адресою, на яку вказує x, на 42 ". Таким чином, попереднє значення x, що дорівнює 23, змінюється на 42.

## Завдання

**Написати процедури, що обчислюють задану функцію за допомогою lambda та let виразів**

**Варіант 1.** Тангенс можна подати у вигляді ланцюгового дробу, де аргумент x дано в радіанах. Визначити процедуру (tan-cf x k), яка обчислює наближення до тангенсу на основі формули ланцюгового дробу:



**Варіант 2.** Кінцевий ланцюговий дріб(finite continued fraction) з k елементі має вигляд:



Визначити процедуру так, щоб обчислення давало значення k-елементного кінцевого ланцюгового дробу.

**Варіант 3 .** За допомогою правила Симпсона інтеграл функції f між **a** і **b** наближено обчислюється у вигляді:



де **h = (b - a) / n**, для якогось парного цілого числа n, yk = f (a + kh). (Збільшення n підвищує точність наближеного обчислення.) Визначити процедуру, яка приймає в якості

аргументів **f, a, b , n,** та повертає значення інтеграла, обчислена за правилом Симпсона.

**Варіант 4.** Напишіть процедуру, яка обчислює добуток значень функції в точках на зазначеному інтервалі. Покажіть, як за допомогою цієї процедури визначити factorial. За допомогою процедури обчисліть наближене значення π за формулою



**Варіант 5.** Розв’язати нелінійне рівняння *x*=cos(*x*) методами перебору та хорд, визначивши інтервал [*a, b*], на якому існує рішення рівняння. Порівняти результати розв’язків двома методами.

**Варіант 6.** Знайти корені нелінійного рівняння виду x=ln(x)+2. Пошук наближеного значення хоча б одного кореня рівняння *f*(*x*) = 0 на відрізку [*a*; *b*] здійснювати методами перебору та дотичних.

**Варіант 7.** Знайти корені нелінійного рівняння виду **x2=e-x**. Пошук наближеного значення хоча б одного кореня рівняння ***f*(*x*) = 0** на відрізку [*a*; *b*] здійснювати методами перебору та хорд

**Варінт 8.** Скласти програму обчислення найбільшого числа Фібоначчі, яке не перевищує число *a* та номера найменшого числа Фібоначчі, яке більше від числа *a*

Числа Фібоначчі обчислюються за рекурентним співвідношенням:



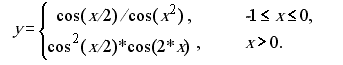
**Варіант 9.** Скласти програму обчислення числа π за формулою Грегорі та за формулою Валліса та порівняти значення

**Варіант 10.** Розв’язати нелінійне рівняння *x*=cos(*x*) методами перебору та хорд, визначивши інтервал [*a, b*], на якому існує рішення рівняння. Порівняти результати розв’язків двома методами.

**Варіант 11.** Скласти програму для знаходження всіх простих чисел із діапазону [2, *n*], які є числами послідовності Фібоначчі

**Варіант 12.** Обчислити значення функції у, розвинувши функцію cos(x) у ряд Тейлора. Аргумент х змінюється від -2 до 2 з кроком 0.5. Визначити похибку.



**Варіант 13**. Обчислити значення функції у, розвинувши функцію sin(x) у ряд Тейлора. Аргумент х змінюється від -2 до 2 з кроком 0.5. Визначити похибку

2_4

**Варіант 14.** Обчислити значення функції у, розвинувши функцію у ряд Тейлора. Аргумент х змінюється від -2 до 2 з кроком 0.5. Визначити похибку.

2_13

**Варіант 15**. Якщо *f* - функція, а *dx* - деяке мале число, то згладжена версія *f* є функція, значення якої в точці *x* є середнє між *f (x - dx), f (x)* і *f (x + dx).* Напишіть процедуру, яка в якості введення приймає процедуру, яка обчислює *f*, і повертає процедуру, яка обчислює згладжену версію *f.*

**Варіант 16.** Якщо f є чисельна функція, а n – додатне ціле число, то n-кратне застосування f визначається як функція, значення якої в точці x дорівнює f (f (... (f (x))...)). Наприклад, якщо f є операція зведення в квадрат, то n-кратне застосування f є функція, яка зводить свій аргумент в 2n степінь. Напишіть процедуру, яка приймає в якості введення процедуру, яка обчислює f, і додатне ціле n, і повертає процедуру, яка обчислює n-кратне застосування f.

**Варіант 17.** Визначте процедуру, яка приймає як аргумент процедуру з одним аргументом і повертає процедуру, яка застосовує вихідну процедуру двічі. Наприклад, якщо процедура **inc** додає до свого аргументу 1, то (**double inc**) повинна бути процедурою, яка додає 2.

**Варіант 18**. Знайти корені нелінійного рівняння виду 

Пошук наближеного значення хоча б одного кореня рівняння *f*(*x*) = 0 на відрізку [*a*; *b*] здійснювати методами перебору та хорд.

# Лабораторна робота 4А. Функції вищих порядків[[1]](#footnote-1)

Deadline жовтень

**Варіант 1.** Побудувати функцію СУМА(Х) через РЕДУКЦІЯ(X,G,A), де Х – список чисел, G – бінарна функція, А – постійний параметр**.**

**Варіант 2.** Побудувати функцію ДОБУТОК(Х) через РЕДУКЦІЯ(X, G, A), де Х – список чисел, G – бінарна функція, А – постійний параметр.

**Варіант 3.** Побудувати функцію СУМА\_ДОБУТОК (Х) через РЕДУКЦІЯ(X, G, A), де Х – список чисел, G – бінарна функція, А – постійний параметр. Результат цієї функції список з 2-ох чисел (добуток, сума).

**Варіант 4.** Побудувати функцію ЗМЕНШИТИ(Х, 2), яка кожен елемент числового списку Х зменшує на 2. Використати попередньо побудовану функцію ВІДОБРАЗИТИ(Х, У), яка застосовує функцію У до кожного елементу списку Х.

**Варіант 5.** Побудувати функцію ЗАЛИШОК(Х, 2), яка визначає залишки від ділення кожного елемента числового списку Х на 2. Використати попередньо побудовану функцію ВІДОБРАЗИТИ(Х, У), яка застосовує функцію У до кожного елементу списку Х.

**Варіант 6.** Побудувати функцію ДІЛЕННЯ (Х, 3), яка визначає частки від ділення кожного елемента числового списку Х на 3. Використати попередньо побудовану функцію ВІДОБРАЗИТИ(Х, У), яка застосовує функцію У до кожного елементу списку Х.

**Варіант 7.** Побудувати функцію ЗБІЛЬШИТИ(Х, 5), яка кожен елемент числового списку Х збільшує на 5. Використати попередньо побудовану функцію ВІДОБРАЗИТИ(Х, У), яка застосовує функцію У до кожного елементу списку Х.

**Варіант 8.** Побудувати функцію ЗБІЛЬШИТИ(Х, 5), яка кожен елемент числового списку Х збільшує в 5 разів. Використати попередньо побудовану функцію ВІДОБРАЗИТИ(Х, У), яка застосовує функцію У до кожного елементу списку Х.

**Варіант 9.** Побудувати функцію ЗМЕНШИТИ(Х, 2), яка кожен елемент числового списку Х зменшує в 2 рази. Використати попередньо побудовану функцію ВІДОБРАЗИТИ(Х,У), яка застосовує функцію У до кожного елементу списку Х.

**Варіант 10.** Побудувати функцію ЗАЛИШОК(Х, 3), яка визначає залишки від ділення кожного елемента числового списку Х на 3. Використати попередньо побудовану функцію ВІДОБРАЗИТИ(Х, У), яка застосовує функцію У до кожного елементу списку Х.

**Варіант 11.** Побудувати функцію ДІЛЕННЯ (Х, 2), яка визначає частки від ділення кожного елемента числового списку Х на 2. Використати попередньо побудовану функцію ВІДОБРАЗИТИ(Х, У), яка застосовує функцію У до кожного елементу списку Х.

# Лабораторна робота 4. Списки та дерева

Deadline жовтень

## Теоретичні відомості

### Поняття та різновиди списків

Основу Scheme, як і Lisp, складають символьні вирази, які називаються S-виразами. (Symbolic expresion). S-вирази – це або атом, або список, або пара.

Атом. Найпростіші об’єкти Scheme, з яких будуються інші структури називаються атомами. Атоми бувають двох типів – символьні і числові.

Символьні атоми будуються з послідовності великих букв та/або цифр, при цьому повинен бути хоча б один символ, відмінний від числа.

*Наприклад*, *SHEME, АВ13, В54, .., 10.А, A..B, p…..p.*

Серед символьних атомів є два спеціальних, які позначають логічні значення істина **– #t** та хибність – #f. Символьний атом розглядається як неділимий цілий об’єкт. До символьних атомів застосовується тільки одна операція: порівняння на тотожність.

Числові атоми – звичайні числа. Наприклад, 124, -344, 4.5, 3.055Е8. Числа – це константи, над якими можна виконувати усі допустимі арифметичні операції та порівняння.

Пара. Об’єднання двох елементів в одне ціле називається парою. Елементами пари можуть бути об’єкти допустимої у функціональному програмуванні структури, тобто атом, пара, список. Пара позначається в дужках, між елементами ставиться крапка, яка розділена з обох сторін пробілами.

*Наприклад*, пара із двох атомів A, B позначається як (А . B). Схематично пара представляється у вигляді комірки поділеної на дві частини 8 (рис. 3). Зауважимо, що S-вираз (A.B), в якому відсутні пробіли з обох сторін крапки є списком, який складається з одного елемента – атома A.B. Аналогічно S-вираз (A..B) – список, який складається з одного атома A..B.

Список. Послідовність зв’язаних між собою елементів, кожен з яких є або атомом, або списком, або парою називають у функціональному програмуванні списком (List). Списки позначаються в дужках, елементи списку розділяються пробілами.

*Наприклад*,

(A D (G H) W) – список з чотирьох елементів: трьох атомів та списку з двох елементів;

(ADW) – список з одного елемента – символьного атома;

(((((FIRST) 2) SECOND) 4) 5) – список з двох елементів: перший елемент – це багаторівневий список, другий - числовий атом. У свою чергу список ((((FIRST) 2) SECOND) 4) містить 2 елемента: багаторівневий список та числовий атом. Вкладений список (((FIRST) 2) SECOND) містить 2 елемента: перший елемент список ((FIRST) 2), а другий елемент – символьний атом SECOND. Цей список ((FIRST) 2) складається з двох елементів: списку (FIRST) та числового атома. (FIRST) – це список з одного символьного атома.

Члени списку організовані в пам’яті комп’ютера у вигляді послідовностей комірок, розділених на дві частини. У першій частині комірки вказується інформація про член списку. Це може бути безпосереднє значення, якщо інформація подається атомом, або адреса комірки, де є перший елемент вкладеного об’єкту, в протилежному випадку. У другій частині комірки вказується адреса зв’язку, тобто адреса знаходження наступного члена заданого списку. Якщо член списку є останнім, то у другій частині комірки вказується символ NIL (“ніщо).

Порожній список позначається ( ).

Неправильно-сформований список. Класичний список містить вкінці вказівник NIL. Натомість існує структура даних, у якій відсутній цей вказівник. Така структура називається неправильно сформованим списком. В графічному представленні, в такій структурі останнім елементом є пара, яка зображена без дужок, наприклад, в структурі ( 1 2 . 3 ) останнім елементом є пара ( 2 . 3 ). Зауважимо, що в такій структурі пара може міститися лише вкінці списку.

Наприклад, правильно-сформований список (1 (2 . 3 ))

Неправильно-сформований список (1 2 . 3)

### Процедури обробки списків

Функція car повертає голову переданого в якості аргументу списку (перший елемент списку називається головою, а залишок списку - хвостом).

Функція cdr повертає хвостову частину списку.

Функція nth повертає елемент списку по його номеру. Ця функція отримує два аргументи: перший - номер елемента в переданому в якості аргументу списку (нумерація починається з нуля); другий - сам список.

Функції first, second, third, повертають відповідно перший, другий, третій, елементи переданого в якості аргументу списку.

Функція last повертає останній елемент списку.

Функція cons будує новий список з переданих їй як аргументи голови і хвоста списку.

Функція list створює список з переданих їй як елементи аргументів. Кількість аргументів довільна.

*Приклад*.

Написати процедуру, яка елементи списку нумерує, починаючи з 1. Для п = 1 процедура повертає перший елемент списку, отриманий в результаті застосування процедури саг, інакше процедура повертає (n-l) -ий елемент списку,отриманий в результаті застосування процедури cdr до вихідного списку.

(define (list-ref items n)

(if (= n 1)

(car items)

(list-ref (cdr items) (- n 1 ))))

(define squares (list 1 4 9 16 25))

(list-ref squares 4)

## Завдання

**Написати процедури, що здійснюють операції над списками та деревами**

**Варіант 1.** Визначити процедуру, яка приймає список як аргумент і повертає список, що складається з тих самих елементів в зворотному порядку. З інвертованого списку отримати список, що містить тільки останній елемент цього (непорожнього) списку.

**Варіант 2**. Напишіть процедуру, яка бере як аргумент дерево, представлене у вигляді списку, і повертає список, елементи якого - усі листя дерева, впорядковане зліва направо.

**Варіант 3**. Множину можна представити як список його різних елементів, а безліч його підмножин як список списків. Наприклад, якщо є множина (1 2 3), то безліч його підмножин складають (() (3) (2) (2 3) (1) (1 3) (1 2) (1 2 3)). Напишіть процедуру, яка породжує безліч підмножин

**Варіант 4**. Для обчислення многочлена



використовується алгоритм Горнера, який многочлен представляє у виді



Обчислення многочлена із змінною x при певному значенні x можна сформулювати у вигляді накопичення. Коефіцієнти многочлена представлені у вигляді послідовності, від a0 до an. Напишіть процедуру, яка обчислює многочлени за схемою Горнера.

**Варіант 5.** Припустимо, що вектори v = (vi) є послідовності чисел, а матриці m = (mij) представлені як послідовності векторів (рядів матриці). Напишіть процедури для обчислення матричних і векторних операцій : Скалярного множення векторів, множення матриці на число

**Варіант 6**. Припустимо, що вектори v = (vi) є послідовності чисел, а матриці m = (mij) представлені як послідовності векторів (рядів матриці). Напишіть процедури для обчислення матричних операцій множення матриці і вектора

**Варіант 7**. Припустимо, що вектори v = (vi) є послідовності чисел, а матриці m = (mij) представлені як послідовності векторів (рядів матриці). Напишіть процедури для обчислення матричних операцій інвертування рядків матриць

**Варіант 8.** Припустимо, що вектори v = (vi) є послідовності чисел, а матриці m = (mij) представлені як послідовності векторів (рядів матриці). Напишіть процедури для обчислення матричних операцій транспозиції матриць

**Варіант 9**. Припустимо, що вектори v = (vi) є послідовності чисел, а матриці m = (mij) представлені як послідовності векторів (рядів матриці). Напишіть процедури для обчислення матричних операцій складання і віднімання матриць

**Варіант 10**. Припустимо, що вектори v = (vi) є послідовності чисел, а матриці m = (mij) представлені як послідовності векторів (рядів матриці). Напишіть процедури для обчислення матричних операцій визначення діагональних елементів матриць

**Варіант 11**. Написати процедуру, яка приймає як аргумент дерево і обчислює добуток квадратів тих з його листків, які є парними числами в діапазоні від 0 до 100

В**аріант 12**. Напишіть процедуру, яка бере як аргумент дерево, представлене у вигляді списку, і повертає кількість вузлів та листків дерева та список, елементи якого є листками дерева з непарними значеннями.

**Варіант 13**. Створити два списки. Якщо перший елемент першого списку є натуральне число, то повернути другий список, інакше повернути список з голови другого й хвоста першого

**Варіант 14**. Задати декілька цілих та дійсних чисел. Написати процедуру, що формує список з максимального й мінімального по модулю чисел, якщо мінімальне й максимальне число - цілі, інакше - середнє арифметичне мінімального й максимального чисел.

**Варіант 15**. Написати процедуру, що для аргументу-списку формує список-результат за правилом: якщо перший й останній елементи списку-аргументу є парні числа, то включити в список-результат квадрати чисел списку-аргументу, інакше сформувати список з першого й останнього елементів.

**Варіант 16.** Написати процедуру, що формує список із заданих користувачем значень. Визначити функцію, яка обертає список і розбиває його на рівні, наприклад:

(a, b, c) = (((c) b) a).

**Варіант 17.** Визначити функцію, яка повертає перетин множин, представлених списками X, Y, що формуються із заданих користувачем чисел.

**Варіант 18**. Визначити функцію, яка повертає різницю множин, представлених списками X, Y, що формуються із заданих користувачем чисел.

**Варіант 19**. Визначити функцію, яка повертає декартовий добуток множин X, Y, ,представлених списками X, Y, що формуються із заданих користувачем чисел і символів.

**Варіант 20**. Визначити функцію, яка групує пари наступним чином:

((a . 1)(b . 2)(c . 3)) => ((a b c) 1 2 3) 7

# Лабораторна робота 5. Обробка текстів

Deadline жовтень

## Завдання

**Написати програму обробки тексту природної мови з використанням функціоналів, що відображають, відповідно до завдання з таблиці. Тексти рекомендується представляти списком списків: кожне речення - список слів, весь текст - список речень.**

**Варіант** 1. Задати текст. Зробити великою першу літеру першого слова кожного речення. Передбачається, що перше слово пропозиції може як починатися, так і не починатися з великої літери.

**Варіант** 2. Задати текст. Зробити великою усі літери кожного слова, що починається з великої літери.

**Варіант** 3. Задати текст. У кожному слові тексту замінити задану користувачем літеру заданою комбінацією літер.

Приклад. Замінна літера: "б", що заміняє поєднання літер: "ку", слово: "абракадабра", результат: "акуракадакура".

**Варіант** 4. У кожному слові видалити літеру, що стоїть між двома заданими.

**Варіант** 5. Сформувати список, що інформує про входження заданої літери в текст у вигляді ((<0 1 5 2 0 >) (<3 0 1 5 2 0 1 0>) ...). Цифри вказують кількість входжень літери в кожне слово речення.

**Варіант** 6. Задати текст. Замінити в кожному реченні все входження заданого слова на заданий нове слово.

**Варіант** 7. Задати текст. Видалити з кожного слова в кожному реченні всі літери, що повторюються.

**Варіант** 8. Задати текст. У кожному слові кожного речення для повторюваних літер зробити наступну заміну: повторні входження літер видалити, до першого входження літери приписати кількість входжень літери в слово.

Приклад: '((Aaabb ccccddd) (eeefggg hhkl)) перетворюється в' ((a3b2 c4d3) (e3fg3 h2kl)).

**Варіант** 9. Задати текст. В кожному слові вставити після заданого 3-літерного поєднання задане 2-літерне.

**Варіант** 10. Задати текст. Вставити задане нове слово після кожного входження іншого заданого слова.

**Варіант** 11. Задати текст. Записати кожне речення тексту в порядку зростання кількості голосних букв в слові.

**Варіант** 12. Задати текст. Переписати кожне речення, розташувавши слова в зворотному алфавітному порядку.

**Варіант** 13. Написати програму, яка в кожному слові вихідного тексту змінює місцями перший і останній літери

**Варіант** 14. "Таємні мови". Побудувати програму перекладу речень української (англійської) мови на так званий циганський жаргон, в якому ключовим словом завжди є наступне слово. Якщо останнє слово залишається без пари, то його можна переводити або в себе, або з деяким заданим допоміжним ключовим словом, наприклад, "плітка".

**Варіант** 15. Написати програму, яка замінює слова вихідного тексту номерами їх семантичних еквівалентов6 по словнику в залежності від значення трьохбуквенні кінці слова. Якщо слово містить менше трьох букв, то заміну не виробляти.

**Варіант** 16. Частотний словарь. Написати програму, яка за заданим текстом будує список пар: (<слово> <частота народження в тексті>).

**Варіант** 17. Написати програму, яка виключає в початковому тексті з кожного слова його закінчення за словником. Словник закінчень представляти списком рядків.

**Варіант** 18. Написати програму, яка в початковому тексті замінює слова, які є значеннями лексичних Функцій (ЛФ) [6] від інших слів того ж тексту, списками виду: {<символ ЛФ за словником> <ключове слово>}. Словник-довідник ЛФ представляти у вигляді списку трійок: (<ключове слово> <символьне позначення ЛФ> <значення ЛФ для ключового слова>). Приклад: ( "дощ" "Magn" "злива").

**Варіант 19.** Написати програму, яка кодує вихідний текст за методом Юлія Цезаря: кожна буква в кожному слові замінюється на наступну

# Лабораторна робота 6. Символьні дані, множини, узагальнені арифметичні операції

Deadline жовтень

## Завдання

**Написати процедури, що здійснюють операції над символьними даними, множинами та узагальненими сeлекторами**

**Варіант 1**. Реалізувати правило отримання похідної:



**Варіант 2**. Визначте узагальнений предикат рівності equ ?, який перевіряє два числа на рівність, і вставте його в пакет узагальненої арифметики з операціями +, -, \*, /. Операція повинна працювати для натуральних чисел, раціональних і комплексних.

**Варіант 3**. Визначте узагальнений предикат = zero ?, який перевіряє, чи рівний його аргумент нулю, і вставте його в пакет узагальненої арифметики з операціями +, -, \*, /. Предикат повинен працювати для натуральних, раціональних і комплексних чисел.

Написати програму символьного диференціювання (по одній змінній) алгебраїчних виразів, представлених у формі правильних префіксних виразів, які містять усі арифметичні операції та вказані у варіанті математичні функції

**Варіанти.**

№4. sin(x), cth(x).

№5. cos(x), pow (x, n).

№6. tg(x), pow (a, x).

№7. ctg(x), ln(x).

№8. exp(x), th(x).

№9. sh(x), arcsin(x).

№10. ch(x), arcos(x).

№11. lg(x), arctg(x).

№12. arcctg(x), ln(x).

№13. pow(x,n), cos(x).

**Варіант 14**. Многочлен є сумою термів, кожен з яких представляє собою коефіцієнт або змінну, що піднесена у степінь, або добуток першого і другого. Два многочлена, над якими здійснюються операції, мають одну й ту саму змінну. Написати процедуру, яка визначає додавання многочленів, використовуючи узагальнену арифметику. Додавання многочленів відбувається по термах. При цьому породжується новий терм того самого порядку, в якому коефіцієнт є сумою коефіцієнтів доданків. Терми одного доданка, для яких немає відповідності в іншому, просто додаються до породжуваного многочлену-сумі.

**Варіант 15**. Многочлен є сумою термів, кожен з яких представляє собою коефіцієнт або змінну, що піднесена у степінь, або добуток першого і другого. Два многочлена, над якими здійснюються операції, мають одну й ту саму змінну. Написати процедуру, яка визначає множення многочленів, використовуючи узагальнену арифметику. Щоб перемножити два списки термів, треба кожен терм з першого списку помножити на всі терми другого. Утворені списки термів накопичуються і утворюють суму. Множення двох термів дає терм, порядок якого дорівнює сумі порядків множників, а коефіцієнт дорівнює добутку коефіцієнтів множників.

**Варіант 16**. Многочлени з однією змінною можна ділити один на одного, отримуючи частку і залишок. Для ділення розділимо старший член діленого на старший член дільника. В результаті вийде перший терм частки. Потім помножимо результат на дільник, віднімемо многочлен, що вийшов з діленого і, рекурсивно ділячи різницю на дільник,

отримаємо решту частки. Зупиняємося, коли порядок дільника перевищить порядок ділимо, і оголошуємо залишком те, що тоді буде називатися діленим. Крім того, якщо коли-небудь ділене виявиться нулем, повертаємо нуль як і частку, і залишок. Розробити процедуру ділення многочленів

**Варіант 17**. Нехай P1, P2 и P3 – многочлени. Нехай Q1 є добутком P1 и P2, а Q2 добутком P1 и P3. Напишіть процедуру, що обчислює найменший загальний дільник (НОД) Q1 и Q2

# Лабораторна робота 7. Програмування многочленів і раціональних функцій

Deadline листопад

## **Завдання**

**Написати процедури, що здійснюють операції над многочленами**

**Варіант 1**. Розробити програму, яка додає та віднімає раціональні функції, які подаються дробами, чисельник та знаменник яких є многочленами, наприклад: (х+1)/(x3+1)

**Варіант 2.** Розробити програму, яка множить та ділить раціональні функції, які подаються дробами, чисельник та знаменник яких є многочленами, наприклад: (х+1)/(x3+1)

**Варіант 3**. Розробити програму, яка скорочує та приводить до найменшого знаменника раціональні функції, які подаються дробами, чисельник та знаменник яких є многочленами, наприклад:



**Варіант 4**. Розробити програму, яка обчислює НОД двух многочленів за алгоритмом Евкліда. Процедура повинна повідомляти про помилку, якщо вхідні об'єкти є многочленами від різних змінних.

**Варіант 5**. НехайP1, P2 и P3 – многочлени, де

P1 : x2 − 2x + 1; P2 : 11x2 + 1; P3 : 13x + 5

Нехай Q1 є добутком P1 и P2, а Q2 - добутком P1 и P3. Написати програму для обчислення НОД Q1 и Q2.

**Варіант 6**. Написати процедуру, яка приймає в якості аргументів два списки термів n і d і повертає список з nn і dd, які представляють собою n і d, приведені до найменшого знаменника. Терм являє собою або коефіцієнт, або змінну,

зведену в степінь, або добуток того та іншого.

**Варіант 7**. Многочлени з однією змінною можна ділити один на одного, отримуючи частку і залишок. Для ділення розділимо старший член діленого на старший член дільника. В результаті вийде перший терм частки. Потім помножимо результат на дільник, віднімемо многочлен, що вийшов з діленого і, рекурсивно ділячи різницю на дільник,

отримаємо решту частки. Зупиняємося, коли порядок дільника перевищить порядок ділимо, і оголошуємо залишком те, що тоді буде називатися діленим. Крім того, якщо коли-небудь ділене виявиться нулем, повертаємо нуль як і частку, і залишок. Розробити процедуру ділення многочленів

**Варіант 8**. Визначити многочлен Р(х) степені N. Задати дійсні числа А та В. Отримати многочлен А\*Р(х-1)-В\*Р(х+1). Реалізувати символьну арифметику обробки многочленів. Многочлен задавати списком

**Варіант 9**. Задати многочлен n-ої степені від одної змінної з цілими коефіцієнтами. Результатом виконання програми має бути розкладання многочлена на множники степені 1 та 2.

**Варіант 10**. Задати многочлен. Утворити алгебраїчну суму, різницю, множення двох многочленів та зведення до цілої степені. Здійснити приведення подібних членів, тобто об'єднувати одночлени, що мають однакові набори змінних, з відповідною заміною коефіцієнтів. Для подання многочленів в пам’яті використовувати списки.

**Варіант 11**. Реалізуйте процедуру factorize для виконання розкладання многочленів виду a2−b2, a3−b3 и a3+b3 за формулами скороченого множення. Процедура приймає один аргумент – вираз, який потрібно розкласти на множники. Піднесення до степеня в початкових виразах реалізувати за допомогою вбудованої процедури expt. Приклади виклику процедури: (factorize '(- (expt x 2) (expt y 2))) ⇒ (\* (- x y) (+ x y))

# Лабораторна робота 8. Обробка списків, черг і таблиць, що змінюються

Deadline листопад

**Завдання: написати процедури, що здійснюють операції над даними, що змінюються, застосовуючи операції присвоєння**

**Варіант 1.** Напишіть процедуру, яка повертає число різних пар в списку. (Підказка: переглядайте структуру, підтримуючи при цьому допоміжну структуру, що стежити за тим, які пари вже були пораховані.) Здійснити видалення елемента за його значенням та за його порядковим номером у списку. У разі відсутності елемента із заданим значенням видати відповідне повідомлення.

**Варіант 2.** Напишіть процедуру, яка розглядає список і визначає, чи міститься в ньому цикл, тобто, не увійде програма, яка спробує дістатися до кінця списку, просуваючись по полях cdr, в нескінченний цикл**.** Якщо циклу немає, то визначити довжину списку і здійснити інверсію списку та пошук елементів, значення яких є числами Фібоначчі.

**Варіант 3**: Побудувати два однозв’язних списків. Порівняти списки. Якщо списки не однакові, виконати їх конкатенацію, інакше утворити новий список, значення елементів яких є парними або непарними числами.

**Варіант 4.** Створити телефонний довідник у вигляді черги, елементами якої є дані про абонента телефонної компанії (прізвище, номер телефону). Здійснити пошук абонента за його прізвищем та за номером. У разі відсутності даних вивести відповідне повідомлення. Надрукувати телефонний довідник у вигляді таблиці.

**Варіант 5.** Створити чергу з чисел. Надрукувати елементи черги. Визначити мінімальне, максимальне значення та їх місцезнаходження в черзі. Знайти середнє арифметичне мінімального та максимального значень черги.

**Варіант 6.** Йде посадка в літак, що розрахований на 50 місць. В черзі на посадку стоять 50 пасажирів. Першим входить старий чоловік. Зайшовши в салон, він сідає на будь-яке випадково вибране місце. Кожний з наступних пасажирів, зайшовши в салон, сідає на своє (позначене в білеті) місце, якщо воно вільне, і на будь-яке з вільних - в протилежному випадку. Яка ймовірність, що останній в черзі пасажир сяде на своє місце?

**Варіант 7.** Сто чоловік стоять у черзі так, що кожен бачить тільки тих, хто стоять перед ним, але чує усе, що говорять інші. На кожну людину одягають ковпак одного з трьох кольорів (скажімо, червоного, синього або білого), колір, зрозуміло, йому невідомий. Починаючи з останнього в черзі (того, хто бачить усіх), у кожної людини запитують, якого кольору у неї ковпак. Якщо вгадує - його залишають в черзі, помиляється - його вилучають. Необхідно придумати, як домовиться цим 100 людям, щоб мінімізувати жертви.

**Варіант 8.** Дек (deque, double - ended queue, "двостороння черга") є послідовністю, елементи в якій можуть додаватися і знищуватися як з голови, так і з хвоста. На деках визначені такі операції: конструктор make - deque, предикат empty - deque?, селекто ры front - deque і rear - deque, і мутатори frontinsertdeque!, rear - insert - deque!, front - delete - deque! і rear - delete - deque!. Покажіть, як представити дек за допомогою пар, і напишіть реалізацію операцій.

**Варіант 9.** Дек - це такий послідовний список, в якому як включення, так і виключення елементів може здійснюватися з будь-якого з двох його кінців. Створити дек з чисел Xj, і=1,...п. Визначити кількість та надрукувати значення елементів дека. Якщо дек має парну кількість елементів, обчислити суму його елементів. Інакше визначити значення виразу



**Варіант 10**. Побудуйте конструктор таблиць make-table, який в якості аргументу приймає процедуру same-key? для перевірки рівності ключів. Make-table повинна повертати процедуру dispatch. через яку можна дістатися до процедур lookup і insert! локальної таблиці.

**Варіант 11.** Узагальнюючи випадки одно- та двовимірних таблиць, покажіть, як можна реалізувати таблицю, в якій елементи зберігаються з будь-якою кількістю ключів і різні значення можуть зберігатися з різною кількістю ключів. Процедури lookup і insert! повинні приймати на вході список ключів, з якими потрібно звернутися до таблиці

**Варіант 12.** При пошуку в таблиці доводиться переглядати список записів. Для великих таблиць може виявитися ефективніше організувати таблицю інакше. Опишіть реалізацію таблиці, в якій записи (ключ, значення) організовані у вигляді бінарного дерева, в припущенні, що ключі можна якимось чином впорядкувати (наприклад, чисельно або за алфавітом).

**Варіант 13**. Мемоізація (memoization) (або табулярізація (tabulation)) - прийом, який дозволяє процедурі записувати в локальній таблиці один раз обчислені значення. Мемоізована процедура підтримує таблицю, де зберігаються результати попередніх викликів, а в якості ключів використовуються аргументи, щодо яких ці результати були отримані. Коли від мемоізованої процедури вимагають обчислити значення, спочатку вона перевірять в таблиці, чи немає там вже потрібного значення, і якщо так, то вона просто повертає це значення. Якщо немає, то вона обчислює значення звичайним способом і заносить його в таблицю. Напишіть мемоізовану версію процедури, що обчислює числа Фібоначчі та заносить їх у таблицю.

# Лабораторна робота 9.

Deadline листопад

# Лабораторна робота 10.

Deadline листопад

# Альтернатива лабораторним роботам з функціонального програмування

Deadline 30 листопада

Виконання тем 1 – 4 забезпечує максимальний бал за семестрову роботу. Може застосовуватися командна робота (3-4 студента в команді)

Одна тема виконується за весь семестр. Відповідно до ітераційної моделі життєвого циклу ПЗ тема ділиться на окремі релізи кількістю до 10, які зараховуються як лабораторні роботи.

1. Аналіз настроїв за відгуками про об’єкт (готель, товар, послугу, тощо)

Теоретичні відомості та підхід до програмної реалізації [2, c.139-164].

1. Логістичне відображення

Теоретичні відомості - <https://en.wikipedia.org/wiki/Logistic_map>

Підхід до програмної реалізації [1, c.291-301]

1. Подання знань в інтелектуальних системах. Фреймова модель та її реалізація на Ліспі (Scheme, Haskell, ect).

Теоретичні відомості [3,4,c. 55-98]

1. Подання знань в інтелектуальних системах. Мережева модель та її реалізація на Ліспі. (Scheme, Haskell, ect).

Теоретичні відомості [4, c. 99-130, 5, c 28-49]

Виконання теми 5 передбачає командну роботу та забезпечує екзамен-автомат з максимальним балом (3-4 студента в команді).

1. **Семантичний аналіз запитів до бази знань інтелектуальної системи ззастосуванням моделей управління предикатних слів**.

Deadline 30 листопада

#### План роботи

#### *Частина 1.*

Розробити мову подання знань відповідно до таких вимог:

- уявлення знань у фреймових одиницях;

- введення ієрархічної структури фреймів, що грунтується на ступені абстракції;

- можливість подання комбінації декларативних і процедурних знань.

Базова структура даних фрейму повинна включати в себе:

- Ім'я фрейму.

- Ім'я слота.

- Покажчики успадкування.

- Вказівка типу даних.

- Значення слота.

- Демон.

- приєднану процедуру.

Необхідно використовувати три способи управління виведенням:

- За допомогою приєднаної процедури - демона;

- За допомогою службової процедури;

- За допомогою механізму спадкування.

#### *Частина 2.*

На основі фреймової моделі розробити і програмно реалізувати систему накопичення знань для конкретної предметної області відповідно до вибраної предметної області (табл. 1). Для подання знань використовувати розроблену в частини 1 мову подання знань.

Таблиця 1. Предметні області для практичної реалізації

|  |  |  |
| --- | --- | --- |
| № п.п | Предметна область | Література |
| 1 | Системи спілкування природною мовою з базами даних. | [6], кн.1, стр. 42-51, 65-94 [6], кн.3, стр. 213-235 |
| 2 | Інтелектуальні запитувально-відповідальні системи. | [6], кн.1, стр. 32-42, [7] |
| 3 | Діалогові системи вирішення завдань. | 6], кн.1, стр. 51-59, [8], [9] |
| 4 | Системи обробки зв'язаних текстів. | [6], кн.1, стр. 59-64 [6], кн.2, стр. 115-126 |
| 5 | Системи мовного спілкування. | [6], кн.1, стр. 95-139 |
| 6 | Системи машинного перекладу. | [6], кн.1, стр. 201-261 |
| 7 | Інструментальні засоби для розробки інтелектуальних систем. | [6], кн.3, стр. 109-168 |
| 8 | Комп'ютерні словники мови (англ, укр., тощо). | [10], [11], [12], [13] |
| 9 | Навчальні системи. | [6], кн.2, стр. 206-231 |

Система повинна складатися з наступних основних модулів:

- Редактор фреймів;

- Модуль перевірки правил;

- Виконавчий механізм;

- Модуль системних функцій.

Редактор фреймів використовується для діалогового побудови фреймових систем.

Модуль перевірки правил служить для виявлення помилок в базі знань, побудованої за допомогою редактора фреймів. Виявлену помилку можна виправити за допомогою того самого фреймового редактора. Прості помилки, що стосуються, наприклад, вказівки успадкування або типу даних, виявляються фреймовим редактором в процесі побудови бази даних, модуль перевірки правил використовується для виявлення більш складних помилок, наприклад, протиріч між фреймами і т.п.

Модуль системних функцій: містить групу функцій, використовуваних для полегшення написання приєднаних процедур.

Класифікація функцій:

a) Функції управління кадрами:

- створення;

- знищення;

- отримання кадру з бази знань;

- отримання типу фрейму;

- отримання списку імен слотів, визначених у фреймі;

- успадкування типу фрейму;

- встановлення підкласу типу фрейму;

- визначення ієрархічної структури бази знань.

b) Функції управління слотами:

- створення;

- знищення;

- отримання слота фрейму;

- отримання значення слота;

- отримання необов'язкового параметра слота;

- установка значення;

- установка необов'язкового параметра.

c) Функція виклику іншого фрейма:

- запуск приєднаної процедури.

d) Функції перевірки:

- перевірка типу даних і їх значень;

- перевірка: визначено чи ні фрейм;

- перевірка реєстрації функції.

e) Інші:

- складання списку одиничної глибини;

- запис в файл;

- загальна частина двох списків.

#### *Частина 3*.

Реалізувати пошукову систему на заданій предметній області.

Пошукова система повинна мати наступну базову структуру:

- база даних (створюється з використанням розробленої в Частини 2 системи накопичення знань).

- редактор створення запиту до бази даних;

- механізм пошуку за запитом (з використанням усіх механізмів виведення на фреймах).

Частина 4.

Доповнити розроблену пошукову систему для користувача інтерфейсом на природній мові.

Вимоги, що пред'являються до багатофункціонального модуля:

- Запит пошукової системи формується користувачем на природній мові (Ділова і наукова проза укр або англ мови).

- Форма запиту - просте поширене запитальне речення укр або англ. мови.

- Зміна бази знань предметної області здійснюється незалежно від підсистеми розуміння природної мови.

- Побудова формального способу сенсу запиту здійснюється кількома етапами (Див. Нижче) відповідно до концепції природної мови як перетворювача "Смисл⇔Текст" [14,15,16]. За основу рекомендується взяти структуру мовної бази знань україномовної або англомовної запитувально-відповідальної системи [14].

- Інтерпретація запиту здійснюється шляхом зіставлення семантичного графа запиту з фреймовой мережею. Сам процес зіставлення є процесом згортання [14] семантичного графа, починаючи від вершин, найбільш віддалених від запитувальної, і закінчуючи запитувальною вершиною. У семантичному графі від запитальною вершини найбільшою мірою віддалені вершини, що не мають дочірніх і мають посилання на вершини фреймовой мережі в своїх словникових статтях. Результатом зіставлення є вичленення контексту, в якому працює запит.

- Відповідь системи формулюється в залежності від характеру запиту [14] у вигляді простого поширеного розповідного речення укр або англ мови.

#### Етапи аналізу вхідної висловлювання:

##### - Морфологічний аналіз.

Вхід: послідовність словоформ вхідного висловлювання.

Вихід: послідовність характеризувалася узагальнених лексем.

При цьому кожна лексема представляється в початковій формі і їй приписується комплекс морфологічних характеристик, що характеризує її. Морфологічний аналіз здійснюється по "зворотного методу" шляхом виділення передбачуваних закінчення і суфікса із застосуванням таблиць афіксів [17,18].

##### - Синтаксичний аналіз.

Вхід: послідовність характеризувалася узагальнених лексем на виході морфологічного аналізу.

Вихід: дерево залежностей, що описує синтаксичну структуру речень.

Синтаксичний аналіз здійснюється на основі методу фільтрів [14] із залученням інформації Моделей Управління [15, 16] предикатних слів. З метою полегшення реалізації на наступному етапі синонімічної перебудови отриманого дерева синтаксичного підпорядкування як відносини підпорядкування рекомендується розглядати відносини глибинного синтаксису [15,16].

- Семантичний аналіз

На етапі Семантичного Аналізу (СЕА) синтаксична структура запиту перетворюється у внутрішнє представлення, що називається Семантичним Графом (СГ).

СГ складається з безлічі вершин-понять, пов'язаних між собою через вершіни-предікати. Дуги СГ мають таке саме значення, як і дуги фреймовой мережі, яка описує знання предметної області. На етапі СЕА йде перебудова дерева синтаксичного підпорядкування, отриманого на етапі синтаксичного аналізу, із застосуванням лексичних і синтаксичних правил синонімічного періфразування (див. [16], стор. 141-176, а також [15], стор. 156-163, 316-345).

Мета зазначеного синонімічного перетворення синтаксичної структури запиту полягає в усуненні порожніх і функціональних предикатів і в перенесення синтаксичних відносин, що виходять з них, на предикат дії [14], стор. 255

## Література

1. Harrop Jon. F# for scientists. - Wiley-Interscience, 2008.
2. Rochester Eric. Mastering Clojure Data Analysis. - Packt Publishing, 2014.
3. . Минский М. Фреймы для представления знаний : Пер. с англ. – М.: Энергия, 1979.
4. Представление и использование знаний. Пер. с япон. / Под ред. Х. Уэно, М. Исидзука. – М.: Мир, 1989
5. Искусственный интеллект. – В 3-х кн. Кн. 2. Модели и методы. – М.: Радио и связь, 1990. С. 28-49
6. Искусственный интеллект : в 3 кн.: Справочник / Под ред. Э.В.Попова. -М.: Радио и связь, 1990
7. Вопросно-Ответные Процессы, Технологии и Системы (Рабочая группа, Ульяновск, Волга, Россия). <http://wg.ulstu.ru/>
8. Койт М.Э. Разработка диалоговой системы с применением корпуса // Компьютерная лингвистика и интеллектуальные технологии: Тр. Междунар. конференции Диалог’2001. Т.2 : Прикладные проблемы - <http://www.dialog-21.ru/>
9. Койт М.Э. Структура информационно-справочного диалога человека с компьютером // Компьютерная лингвистика и интеллектуальные технологии: Тр. Междунар. конференции Диалог’2003.- <http://www.dialog-21.ru/>
10. Машинный фонд русского языка : идеи и суждения : [Сборник] / АН СССР, Институт русского языка, Научный совет по лексикологии и лексикографии; Отв. Ред. Ю.Н. Караулов. – М.: Наука, 1986
11. RussNet : тезаурус русского языка. - <http://www.phil.pu.ru/depts/12/RN/index_ru.shtml>
12. Леонтьева Н.Н. Русский общесемантический словарь (РОСС) : структура, наполнение. // НТИ. Сер. 2. 1997. №12. С. 5-20.
13. АОТ : Автоматическая Обработка Текста. Рабочая группа Aot.ru. // <http://www.aot.ru/>
14. Попов Э.В. Общение с ЭВМ на естественном языке. - М.: Наука, 1982. - 360 с.
15. Апресян Ю.Д. Избранные труды. Т.I. Лексическая семантика. Синонимиче-ские средства языка - М.: Школа "Языки русской культуры", 1995. - 472 с.
16. Мельчук И.А. Опыт теории лингвистических моделей "СмыслЫТекст" : Семантика, синтаксис - М.: Школа "Языки русской культуры", 1999. - 345 с.
17. . Белоногов Г.Г. и Богатырев В.И. Автоматизированные информационные системы. Под ред. К.В. Тараканова. - М.: Сов. радио, 1973. - 328 с.
18. Белоногов Г.Г., Новоселов А.П. Автоматизация процессов накопления, по-иска и обобщения информации. - М.: Наука, 1979. - 256 с.

1. Лабораторна робота 4А виконується на додаткові бали [↑](#footnote-ref-1)