УДК 004.43
DOI: 10.18523/2617-3808.2020.3.17-26

Ющенко Ю. О.

ОКРЕМІ АСПЕКТИ ДЕКЛАРАТИВНОСТІ «МІНУС ШТРИХ-ОПЕРАЦІЇ»

У роботі розглянуто «мінус штрих-операцію», яку було введено як обернену до «штрих-операції» (1955 р.). Аналогом «штрих-операції» є розіменування вказівника (1964 р.).

На прикладі продемонстровано, що для отримання адреси лінійного однозв'язного списку в Адресному програмуванні можна вказати порядковий номер цього вузла у списку. У цьому полягає перевага «штрих-операції» над розіменуванням вказівника.

«Мінус штрих-операцію» в імперативних мовах програмування високого рівня не застосовують, оскільки за своєю сутністю вона належить до декларативної концепції програмування.

У роботі досліджено динамічну (неявну) типізацію даних в Адресній мові програмування та наведено загальну класифікацію типів даних.

У статті подано приклад представлення дерева в Адресній мові програмування шляхом указування «батьків» вершин дерева, без зазначення сукупності синів, як це потрібно робити в імперативних мовах програмування. В Адресному програмуванні сукупність адрес усіх синів можна отримати застосуванням до адреси вершини «мінус штрих-операції». Із цього та інших наведених прикладів випливає універсальна потужність «мінус штрих-операції» як інструменту декларативного програмування.

Адресна мова програмування є багатоконцептуальною та поєднує у собі концепції імперативного та декларативного програмування. Основою декларативної концепції Адресного програмування (1955 р.) є «мінус штрих-операція».

Ключові слова: «штрих-операція», «мінус штрих-операція», адресне сортування, багатовимірне адресне сортування, списки, двозв'язні списки, дерева, абстрактні типи даних, структури даних, прості дані, скалярні дані, масиви, записи, вказівники, показники, посилання, Pointers, типи даних, неявна типізація, неявна типізація даних, бази даних, реляційні бази даних, SQL, Foreign Key, інформаційні системи, логічне програмування, імперативне програмування, декларативне програмування, мови програмування.

Сучасні мови імперативного програмування неможливо уявити без вказівників та операції їх розіменування (англ. dereference operator), яку вперше запропонувала Катерина Ющенко в Адресній мові програмування (1955 р., м. Київ) [2; 1; 3]. Оскільки у 1950–1960-х рр. існувала т. зв. залізна завіса між соціалістичними та капіталістичними країнами, винахід вказівників та їх застосування помилково приписують Гарі Лаусону (1964 р.).

В основу Адресного програмування покладено «принцип адресності», який у стислому вигляді сформульовано таким чином: *«У програмі вказують не власне числа, а адреси комірок пам'яті, ініціалізовані потрібними значеннями»* (під значеннями розуміють числові значення різного типу, довільні адреси даних та адреси функцій/підпрограм) [2]. Назвемо це **тезою № 1 принципу адресності**. В окремому розділі «Принцип адресності» підручника з програмування [2] надано розширені пояснення цього принципу, визначено адресні та інші функції Адресного програмування та зазначено універсальність цього принципу для довільного групування даних і підпрограм. До найпростіших, базових адресних функцій належать відома «штрих-операція» (розіменування вказівника) та арифметичні операції з вказівниками: додавання та віднімання від значення адреси (вказівника) цілих чисел.

Для подальшого розкриття принципу адресності потрібно згадати загальновідомі відомості щодо використання пам'яті.

Пам'ять комп'ютера (як оперативна, так і зовнішня) є лінійною: послідовність значень адрес комірок пам'яті, що починається з адреси: **«0 ... 00»** та закінчується **«11 ... 1»** (тут кількість **«0»** та **«1»** визначається адресною шиною комп'ютера).

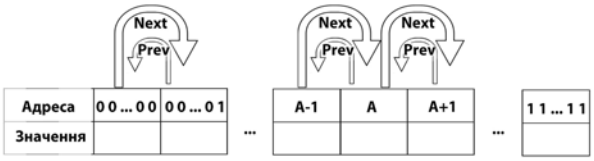

Рис. 1. Лінійність пам'яті комп'ютера та відношення слідування: «Next» і «Prev»

Кожна адреса **«A»**, окрім першої та останньої, має наступну адресу **«A+1»** та попередню адресу **«A-1»**. Таким способом на всіх адресах комп'ютера визначено бінарне відношення слідування «Next» та обернене до нього відношення «Prev». На рис. 1 проілюстровано лінійність пам'яті комп'ютера та відношення слідування.

Отже, усі дані, зокрема програми та підпрограми, пов'язані між собою відношенням слідування у пам'яті комп'ютера, яке може бути використано задля групування даних, що зберігаються за адресами. Варто зазначити, що *n*-кратне «застосування» відношення «Next» («Prev») дає змогу адресуватися (звертатися) до даних «*n*-наступний» («*n*-попередній»). За $n = 2$ це: «позанаступний» (допопередній»).

Найпростішим широковідомим групуванням даних (змістів адрес), яке використовує слідування адрес пам'яті комп'ютера, є групування даних у масиви. Позначимо адресу масиву з *n* елементів (адресою його першого елементу) літерою *M* (базою). Зауважимо, що адреса першого елементу масиву збігається з адресою масиву *M*. На рис. 2 зображено принцип групування даних у масиви та спосіб адресації (звернення) до значень елементів масиву з *n* елементів. У цьому прикладі припущено, що зміст адреси (значення елементів масиву) займає одну комірку пам'яті комп'ютера.

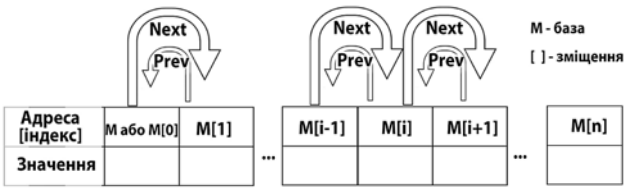

Рис. 2. Групування даних у масиви

Звернення (адресацію) до *i*-го елементу масиву **M[i]** називатимемо «стрибком на i» («стрибком вперед через *i-1* адресу»). У таких випадках прийнято *i* називати зміщенням від бази **M**. Запис **M[i-j]** назвемо «стрибком від i назад на j».

Якщо збереження елементу масиву потребує *k* комірок пам'яті, то звернення до *i*-го елементу масиву відбувається таким способом: **M[i*k]** або **M+i*k**.

Аналогічним чином відбувається звернення (адресація) до елементів багатовимірних масивів. Наприклад адресація до *(i, j)*-го елементу двовимірного масиву відбувається так: **M[i, j]** або $\mathbf{M+i*k+j*k*n_1}$, де n_1 – кількість (розмір) першого виміру масиву. Запис $\mathbf{M+i*k+j*k*n_1}$ являє собою «стрибок на $\mathbf{i*k+j*k*n_1}$».

Можливість цих стрибків дає змогу створювати зв'язки між даними, які вже не є лінійними. Найбільш яскравим прикладом створення нелінійного зв'язку з використанням слідування, яке визначено лінійністю пам'яті комп'ютера, та «стрибків» є широковідоме моделювання урівноважених *n*-арних дерев у одновимірному масиві.

На рис. 3 подано приклад моделювання бінарного урівноваженого дерева з дев'ятьма елементами. Вузли дерева позначено індексами одновимірного масиву **M**. Значення вузлів дерева зберігаються в елементах масиву **M[i]**, де і пробігає значення від *1* до *9*. Для спрощення формул приймаємо, що першим елементом масиву є елемент номер *1:* **M[1].**

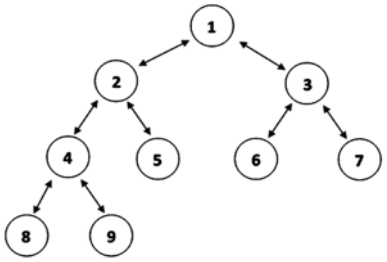

Рис. 3. Приклад моделювання в одновимірному масиві бінарного урівноваженого дерева з дев'ятьма вузлами

За номером (індексом масиву) *k* довільного вузла, окрім кореня, можна визначити індекс *f* батька з формулою: **f=[k/2]** (тут квадратні дужки «[ ]» означають цілу частину від ділення).

Для визначення синів вузла з індексом *k* використовують формули:

LeftSon=k*2 і **RightSon=k*2+1**

Таке бінарне дерево є нелінійною структурою. Масиви можна використати для створення інших зв'язків між даними.

Наведемо ще один приклад створення лінійного слідування, яке не збігається з лінійним слідуванням адрес пам'яті комп'ютера.

Цей приклад стосується так званого адресного сортування даних [4; 5]. Під *адресним сортуванням* розуміють створення структури даних, яка дає змогу звертатися до даних у порядку їх зростання (спадання) за збереження місць розташування самих даних у пам'яті комп'ютера. Зауважимо, що адресне сортування вперше в світі було реалізовано декількома різними способами в адресній мові програмування [2; 1; 3].

Нехай у нас є масив **M** із *n* числовими даними. Результати адресного сортування за зростанням

можна зберегти в масиві індексів (або індексному масиві) **Mi**. У масиві **Mi** зберігаються цілі числа, які є індексами масиву **M**. Масив **Mi** зберігає результати адресного сортування за зростанням елементів масиву **M**, якщо: **M[Mi[i]] >= M[Mi[j]]** для будь-яких значень *i* та *j*, таких, що **i>=j**. Очевидно, що при виконанні цієї умови елемент **M[Mi[1]]** є найменшим елементом масиву **M**, а елемент **M[Mi[n]]** є найбільшим елементом **M**. Ми не розглядаємо методи побудови масиву **Mi**, оскільки не принципово, який метод сортування було застосовано. Можна було використати метод бульбашкового сортування, сортування розчином, сортування злиття та будь-який інший метод сортування даних, зокрема алгоритми швидкого сортування.

Очевидно, що *i*-м за зростанням елементом масиву **M** є елемент **M[Mi[i]]**. Отже, слідування елементів масиву **Mi** збігається зі слідуванням елементів масиву **M** у зростальному порядку та не ґрунтується на використанні лінійного слідування адрес пам'яті комп'ютера. Результати адресного сортування продемонстровано на рис. 4.

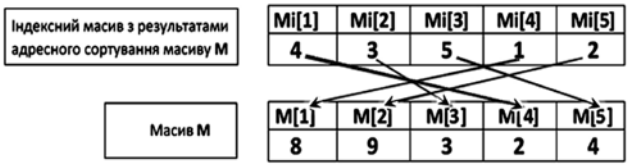

Рис. 4. Приклад результатів адресного сортування

Якщо елементами масиву **M** є не скалярні дані, а згруповані дані, як-от рядки, кортежі чи екземпляри класу, то можна побудувати не один масив індексів, а декілька. Кожен із масивів індексів у такому випадку зберігатиме результати сортування елементів масиву **M** за однією зі складових (рядком, кортежем, даними класу). Технологія збереження результатів сортування одразу за декількома складовими згрупованих даних отримала назву *багатовимірне адресне впорядкування*. Додатковою та обов'язковою умовою багатовимірного адресного впорядкування є можливість переходу від довільного елемента даних до наступного чи попереднього за одиницю умовного часу. Потрібно зазначити, що побудова декількох масивів індексації не являтиме собою багатовимірне адресне сортування, оскільки не буде забезпечено умову «швидкого» переходу до наступного (попереднього) елементу в довільному вимірі. Для підкорення цієї умови необхідно створити так звані масиви *зворотної індексації*, які дають змогу за порядковим номером елементу масиву визначити його порядковий номер у кожному з вимірів. Побудова масивів зворотної індексації вельми проста та входить до кола питань цієї роботи.

Нехай у нас є два масиви не скалярних даних (рядків, кортежів або екземплярів об'єктів) **A** та **B**.

Структури даних, до складу яких входять функції (їхні адреси), називатимемо *«об'єктами»*. Це слово візьмемо в лапки, щоб розрізняти із поняттям об'єкта в об'єктно-орієнтованому програмуванні (ООП).

Аналогічним способом, як і з масивами індексації, між даними цих масивів, як таблицями реляційної бази даних (РБД), можна визначити відношення «1 до *n*». Можна до складу атрибутів масиву додати ще один атрибут, який заповнити значенням так само, як це роблять у РБД (вказати значення індексів масиву **B**).

Для подання відношення «*n* до *m*» між елементами масивів **A** та **B** можна використати двовимірний проміжний масив аналогічно тому, як це роблять у РБД із визначенням проміжної таблиці. У першому стовпчику можна зберігати «зовнішні» ключі (індекси) до масиву **A**, а другий стовпчик використати для зовнішніх ключів до масиву **B**. При цьому, рядками проміжного масиву будуть елементи відношення «*n* до *m*» між масивами **A** та **B**.

Наведені вище приклади були надані для демонстрування доволі широких можливостей мов програмування, в яких немає вказівників (Pointers). У перших імперативних мовах програмування високого рівня FORTRAN (1958 р.), COBOL (1959 р.) та ALGOL-60 (1960 р.) вказівників не було.

Продовжимо розглядати принцип адресності.

Згідно з термінологією Адресного програмування змінні у мовах програмування високого рівня використовують адресацію першого рангу, під якою розуміють разове, однократне застосування «штрих-операції», яку визначають як отримання даних за певною адресою. В Адресному програмуванні однократне застосування «штрих-операції» називають *прямою адресацією*, або *адресацією першого рангу*. При отриманні значень елементів масивів, складових елементів записів, як і значень скалярних змінних, використовують пряму адресацію.

Принцип адресності передбачає, що за певними адресами пам'яті є можливість зберігати не лише скалярні дані (значення), а й адреси в пам'яті комп'ютера. Цю тезу назвемо тезою № 2 принципу адресності.

Ця теза має важливе значення та надає можливості використання непрямої або опосередкованої (*рос.* косвенной) адресації та адресації вищих рангів. Фактично ці можливості повністю покривають можливості вказівників, які запропонував Гарі Лаусон у мові PL/1 у 1964 р. «Штрих-операція» має певні переваги порівняно із розіменуванням вказівника (див. таблицю).

Таблиця. **Відповідність «штрих-операції» і операції розіменування вказівника**

Коментар (ранг адресації, особливі умови)	**Адресна мова (1955 р.)**	**Pascal**	**C**
Адресація першого рангу	‘A	A^	*A
Адресація другого рангу	^{2}A або ‘‘A або ‘(‘A)	A^^	**A
Адресація третього рангу	^{2}A або ‘A або ‘‘A	A^^^	***A
Адресація *n*-го рангу	^{n}A	можна реалізувати з циклом	
Функція як аргумент операції	‘f()	f^	*f()

Повний семантичний збіг «штрих-операції» та операції розіменування вказівника випливає з визначення «штрих-операції»: *отримання значення за адресою*. «Штрих-операцію» позначають штрихом: «‘».

Теза принципу адресності № 2 дає змогу визначати відношення слідування, яке може бути нелінійним, та пов'язувати між собою довільні дані, зокрема раніше згруповані дані. Потрібно нагадати, що в Адресному програмуванні значеннями за адресами можуть бути змістовні дані (скалярні) та незмістовні дані (адреси). Змістовними даними в Адресному програмуванні можуть бути скалярні дані, а адреси можуть вказувати як на згруповані дані (довільні структури), так і на адреси функцій.

Спочатку розглянемо приклади групування даних в Адресному програмуванні без застосування непрямої (опосередкованої) адресації та адресації вищих рангів. Для порівняння буде надаватися відповідність групування даних, яку використовують в імперативних мовах програмування високого рівня.

Із наведених вище прикладів випливає, що групування скалярних даних у масиви в Адресній мові програмування здійснюється тим самим способом, що і в інших мовах програмування високого рівня. Єдиною відмінністю Адресної мови програмування від інших мов програмування є можливість використання синтаксичної конструкції **‘(M+i)** для звернення (адресації) до *i*-го елементу масиву **M**.

В Адресній мові програмування є синтаксичні відмінності здійснення адресації до складової частини (поля) згрупованих скалярних даних різного типу. За адресації до певної складової в Адресній мові вказують її порядковий номер (номер позиції входження у запис). Продемонструємо це на прикладі. Нехай є запис (у розумінні мови Паскаль) типу:

```
type e = record
     age, weight, quantity: integer
end;
var a: example;
```

На Паскалі звернення до поля **age** змінної **a** синтаксично записується так: **e.weight**. В Адресній мові до адреси **e** потрібно таким способом застосувати адресні функції: **‘(e+1)**, а для адресації до поля: **quantity ‘(e+2)**. На рис. 5 продемонстровано ці звернення.

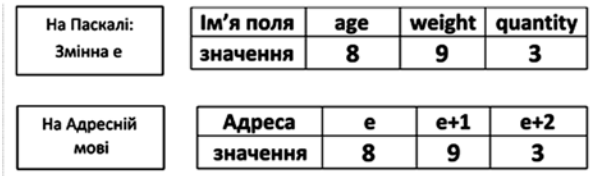

Рис. 5. Приклад групування скалярних даних (аналог запису у Паскалі) без використання адресації вищих рангів

Потрібно зауважити, що в Адресній мові є можливість наблизити синтаксис звернення до окремих полів до синтаксису в інших мовах програмування шляхом визначення простих функцій, імена яких можуть бути мнемонічними позначками. Наприклад, можна визначити функцію **weight(e),** що повертає адресу, яка безпосередньо йде за адресою **e**. Для поля **age** можна визначити тотожну адресну функцію з цим іменем: функція повертає ту саму адресу, яка була отримана при виклику. Запис адресації в Адресній мові «**weight(e)**» стає схожим на запис на Паскалі: «**e.weight**».

«Об'єкти» в адресному програмуванні

Використання адресації вищих рангів в Адресній мові дає можливість створювати об'єктно подібні об'єднання даних із методами (функціями) їх оброблення. На рис. 6 наведено приклад аналогу екземпляру класу («об'єкт») паралелепіпед, в якому визначено три сторони та функцію визначення його об'єму.

Адреса	e	e+1	e+2	e+3
значення	V	9	3	2

Тут V – ім'я функції (або адреса, що є тим самим), яка визначає об'єм паралелепіпеду

Рис. 6. Приклад «об'єкта» паралелепіпед в Адресному програмуванні

Для визначення обсягу паралелепіпеда **e** (паралелепіпеда за адресою **e**) Адресною мовою має бути записано таке: **'e(e+1, e+2, e+3)**. Цим записом визначається, що викликається функція, яка розташована за адресою, яку вказано як значення адреси **e**, а як параметри цього виклику виступають три значення, які містяться за адресами: **e+1, e+2** та **e+3**. Цей запис не зовсім лаконічний, але, використавши вищенаведений прийом, можна визначити функцію з іменем **vol** із одним аргументом **e,** в тілі якої буде записано один оператор засилання: **'e(e+1, e+2, e+3) => vol**. В результаті звернення до функції **vol(e)** відбудеться засилання значення, підраховане за формулою **'(e+1)*'(e+2)*'(e+3),** визначення обсягу паралелепіпеда не за адресою розташування функції **vol**, а *в «місце, де записано виклик функції».* Тут ім'я функції є своєрідним носієм значення, яке ця функція підраховує, *хоча ім'я функції являє собою адресу її розташування в пам'яті комп'ютера.* Це твердження жодною мірою не суперечить поняттям імперативних мов програмування високого рівня.

В Адресному програмуванні можна визначати адресні функції, які виділяють нову пам'ять (адреси) для збереження значень за цими адресами. При цьому відбувається виділення пам'яті для усіх складових, які включено до області доступності щодо адреси, під яку виділяється пам'ять. Тобто виділення пам'яті відбувається відповідно до типу даних, на яку посилається адреса. Ми застосували термін «тип даних», який ще не було визначено. Тепер необхідно визначити поняття типу даних в Адресній мові програмування.

Типи даних в Адресному програмуванні

В Адресному програмуванні, як і в мові програмування **Python**, використовують неявну типізацію даних. За кожною адресою зберігається значення певного типу, але оголошувати тип значення за адресою не потрібно. При засиланні значення за адресою «система» самостійно визначає тип цього значення.

За будь-якою адресою пам'яті комп'ютера в Адресній мові можуть зберігатися значення. Цими значеннями можуть бути:

а) *скалярні значення*: числові, булеві та інші типи (можна припустити, що в Адресній мові є типи значень, які використовують у сучасних мовах програмування, хоча цих типів значень у 1955 р. ще не існувало);

б) *адреси*.

Зі скалярними значеннями все прозоро, оскільки жодних відмінностей із мовами програмування високого рівня немає.

Якщо за адресою (первинною адресою) зберігається інша адреса, то для визначення типу цієї адреси потрібно визначити область доступності цієї первинної адреси.

Центральне місце в Адресній мові посідає поняття області доступності, яке визначається щодо всієї програми, її фрагмента (сукупності операторів) та щодо окремої адреси. *Область доступності певної* адреси – це множина усіх адрес, до яких може здійснюватися доступ шляхом застосування адресних функцій.

Під типом адреси розумітимемо її область доступності.

Адреси можуть посилатися на масиви (їхні окремі елементи) скалярних значень, на масиви адрес та на інші, довільним чином згруповані, сукупності (структури) даних.

Масиви мають певну визначену розмірність (одновимірний, двовимірний або *n*-вимірний) із визначеними розмірами щодо кожного виміру. Інші складені сукупності даних у пам'яті комп'ютера представлені як лінійними списками, так і такими, що мають довільну структуру: циклічні списки, дерева, графи тощо. Складені сукупності даних як свої елементи можуть мати скалярні дані, інші сукупності скалярних даних, сукупності складених даних, сукупності складених даних із функціями («об'єкти») та сукупності рекурсивно визначених даних.

До адресних функцій, які визначають область доступності, належать арифметичні операції над адресами, «штрих-операція», «мінус штрих-операція» та інші адресні функції, які визначені в адресній програмі.

Арифметичні операції над адресами (адресами елементів масивів) є простими та очевидними. Якщо адресою є адреса масиву або адреса певного його елементу, то до області доступності належать усі адреси елементів масиву. Система визначає допустимість і недопустимість виконання арифметичних операцій над адресами (додавання до адреси та віднімання від неї цілих чисел), які входять до складу масивів.

Базову адресну функцію «штрих-операція» над адресою (розіменування вказівника) було визначено вище.

Нижче буде визначено ще одну базову адресну функцію – «мінус штрих-операцію», яка є оберненою (зворотною) до «штрих-операції».

Програмісти мають можливість визначати власні адресні функції шляхом застосування суперпозиції (або композиції, або підстановки) інших, раніше визначених адресних функцій і не лише їх. При визначенні адресних функцій можна використовувати інші функції Адресної мови програмування, як-от *предикатні функції* (аналог предикатів у розумінні логічного програмування); *оператори засилання значень* (повний аналог оператора присвоювання); розгалуження програми, можливість якого в Адресній мові надається так званими *розпізнавачами* (аналоги умовних операторів та умовних виразів). Розпізнавачі в Адресній мові слугують як для розгалуження операторів програми, так і для розгалуження при обчисленні значень виразів (умовні вирази). При визначенні адресних функцій можна використовувати усі конструкції та засоби Адресного програмування, до яких належать і *теоретико-множинні операції над множинами*. В Адресній мові є *булеві операції*, які застосовують в умовах (логічних виразах) розпізнавачів. При визначенні «нових» предикатних функцій можна використовувати, як аргументи булевих операцій, раніше визначені предикатні функції.

Потрібно звернути увагу на дуже важливу обставину: *область доступності є динамічною та може змінюватись під час виконання програми.* Область доступності визначається станом пам'яті – бінарним відношенням (адреса, значення). Оператори засилання *«вираз» => «адресний вираз»* змінюють стан пам'яті шляхом заміни відповідного елементу стану пам'яті на елемент: «*адреса, яка збігається з визначеним значенням адресного виразу», «підраховане значення виразу».* Якщо значення скалярних даних змінюється на інше скалярне значення (наприклад, оператором засилання), то область доступності залишається без змін. Але якщо змінюється значення, яке є адресою, на інше значення адреси, то область доступності може змінитися.

Тепер можна перейти до розгляду прикладів типів даних (адрес) Адресного програмування. Приклади скалярних типів даних не наводитимемо, оскільки вони нічим не відрізняються від типів даних у інших мовах програмування.

Приклади типів даних Адресного програмування

Спочатку розглянемо приклади типів даних, у яких немає застосування «мінус штрих-операції». Приклади деяких типів даних вже було наведено вище. Зокрема, на рис. 2 – приклад *типу даних масиву цілих чисел.* Усі адреси елементів масиву мають однакову область доступності, якою є сукупність адрес усіх елементів масиву. Наприклад, якщо всі елементи масиву мають тип цілих чисел, то типом адрес є масив цілих чисел (усі адреси його елементів). Адреса першого та будь-якого іншого елементу масиву скалярних даних має як тип даних (області доступності) сукупність (або множину) адрес усіх елементів масиву.

На рис. 5 наведено тип даних, який відповідає *записам* у Паскалі та реалізується за тим самим принципом, що і масив. Із погляду Адресного програмування записи Паскаля являють собою «масиви», елементи яких набули різні типи скалярних даних.

На рис. 6 подано *тип даних «об'єкт»*, оскільки до його складу входить адреса, значенням якої є адреса функції.

Перейдемо до розгляду прикладів типів даних із використанням непрямої адресації та адресації вищих рангів.

Приклад лінійного однозв'язного списку

На рис. 7 показано лінійний однозв'язний список, а також наведено його порівняння зі списками мов програмування високого рівня, в яких існують вказівники (**Pointers**). Цей список містить *n* вузлів із цілими числами в них.

Рис. 7. Приклад лінійного однозв'язного списку

Із поданого прикладу випливає очевидна перевага списків в Адресній мові програмування порівняно зі списками, створеними з використанням вказівників, оскільки в Адресному програмуванні є можливість прямого звернення (адресації) до інформації в довільному вузлі за його порядковим номером. Можливість звертатися до вузлів списку так само, як і до елементів масиву, є безсумнівною перевагою Адресної мови програмування. Зокрема, для отримання інформації, яка збережена в *i*-му вузлі, достатньо записати: **'(ⁱList+1)**.

У розглянутому прикладі у вузлах списку зберігаються прості, скалярні дані (цілі числа), але, так само, як і в мовах зі вказівниками, у вузлах списку можуть бути групи даних. Особливістю списків Адресної мови програмування є те, що в ній допускається збереження різних типів даних у різних вузлах списку.

Із цього прикладу має бути очевидною можливість представляти в Адресній мові дерева довільної арності, будь-які дерева та інші довільні абстрактні типи даних.

Потрібно зазначити, що Адресна мова програмування має дещо ширші можливості представляти абстрактні типи даних. Продемонструємо це на прикладі дерев із довільною та невизначеною кількістю синів. У сучасних імперативних мовах програмування такі дерева подаються вузлами, в яких зберігаються дані вузла та список синів. В Адресній мові можна представляти дерева точно так само. При цьому, на відміну від сучасних імперативних мов програмування, доступ до *i*-го сина може здійснюватися за його номером, а не «мандруванням» по списку синів: якщо **S** – адреса першого сина деякого вузла, то $^{i-1}$**S** – є адреса *i*-го сина цього вузла.

В Адресній мові є й інші можливості представляти такі дерева. Першою такою можливістю є визначення в кожному вузлі масиву синів. При цьому кількість елементів масиву синів у кожному вузлі збігатиметься з кількістю синів у цьому вузлі.

Нижче буде продемонстровано спосіб представлення дерев із застосуванням «мінус штрих-операції». У разі використання цієї операції зникає потреба вказувати для кожного вузла перелік його синів. У сучасних імперативних мовах програмування немає аналогів «мінус штрих-операції», що унеможливлює представлення у них дерев у такий спосіб.

Перед тим, як визначити «мінус штрих-операцію» та продовжити визначати типи даних Адресного програмування із наданням прикладів типів даних, зупинимося на класифікації мов програмування за *«принципом адресності» за трьома ступенями* [2], що дасть змогу ввести відповідну класифікацію типів даних в Адресній мові програмування.

Класифікація мов програмування за принципом адресності

Згідно з принципом адресності, мови програмування розподіляють на три ступені [2].

До *першого ступеня* відносять мови програмування, в яких використовують лише адресацію першого рангу, а непряму адресацію, адресацію другого рангу та адресацію вищих рангів не застосовують. Іншими словами, за адресами в пам'яті комп'ютера можуть зберігатися лише прості, скалярні дані. Масиви в цих мовах реалізуються шляхом використання лінійності пам'яті комп'ютера. В комірках (змінних) не можуть зберігатися адреси. Тобто до мов програмування першого ступеня належать мови програмування без вказівників, зокрема, перші закордонні мови програмування Фортран (1958 р.), Кобол (1959 р.) та Алгол-60 (1960 р.). Неможливість адресації вищих рангів обмежує типізацію в таких мовах програмування, у них існують лише скалярні змінні, масиви однотипних скалярних змінних і записи. При цьому потрібно зауважити, що записів не було ані в мові Фортран, ані в мові Алгол-60. Однак із використанням масивів і записів у цих мовах програмування є можливість модулювати інші, корисніші структури даних.

Приклад модулювання одновимірним масивом бінарного врівноваженого дерева надано на початку цієї роботи та проілюстровано на рис. 3. Було також показано можливість використання списків для визначення зв'язків між даними «**1 до n**» та «**n до m**». На рис. 4 наведено приклад методу визначення відношення лінійного слідування, яке розбігається з лінійним слідуванням адрес у пам'яті комп'ютера, на якому базується реалізація типів даних масиву.

До *другого ступеня* належать мови програмування, в яких, окрім використання прямої адресації, використовують непряму адресацію, адресацію другого рангу та адресацію вищих рангів. Вказівники являють собою частковий випадок принципу адресності, а операція розіменування вказівника повністю покривається «штрих-операцією». Отже, до мов програмування другого ступеня належать мови програмування з вказівниками та з операцією їх розіменування. Слід зазначити, що без операції розіменування вказівників вказівники були б некорисними.

Зауважимо, що класифікацію мов програмування за трьома ступенями було опубліковано у 1961 р. [1; 2], і на той час, окрім Адресної мови програмування, у світі ще не було інших мов програмування другого ступеня. Першою невітчизняною мовою програмування другого ступеня стала мова PL/1, до синтаксису якої Гарі Лаусон у 1964 р. запропонував ввести вказівники (Pointers). Згодом з'явилися мови програмування другого ступеня, в яких немає вказівників у явному вигляді, а їх наявність «прихована» від програмістів.

Згідно з принципом адресності, наявність «мінус штрих-операції», предикатних функцій і теоретико-множинних операцій над множинами в мовах програмування визначає належність їх до мов програмування *третього ступеня*.

За принципом адресності, декларативні мови програмування належать до мов третього ступеня.

Аналогічно до класифікації мов програмування в [1] введено класифікацію програм на програми першого, другого та третього ступенів, залежно від використання або невикористання в них засобів адресації: вищих рангів та «мінус штрих-операції».

Так само можна застосувати класифікацію типів даних за трьома ступенями.

До *першого ступеня* належать типи даних, у яких доступ до значень складових частин передбачає лише пряму адресацію (адресацію першого рангу) та арифметичні операції з адресами. Це дані мов програмування, у яких немає вказівників: прості, скалярні дані, масиви та записи (як-от у мовах Кобол і Паскаль). Типи даних першого ступеня називатимемо *примітивними типами даних*.

Складніші типи даних не можна реалізувати без наявності вказівників (у явному чи прихованому від програмістів вигляді). До *даних другого ступеня* віднесемо усі дані, які можуть використовувати доступ (адресацію) із застосуванням не лише прямої, а й непрямої адресації (адресації вищих рангів). До таких типів даних, наприклад, належать структури, які містять підструктури, масиви скалярних даних і підструктур, об'єкти (екземпляри класів), списки, дерева та інші абстрактні типи даних у розумінні концепції імперативного програмування. Типи даних другого ступеня називатимемо *структурними типами даних*. Доступ (адресація) до складових структурних типів даних здійснюється з використанням прямої, непрямої адресації вищих рангів та арифметичних операцій над адресами. Із наведеного визначення випливає, що структурні типи даних містять примітивні типи даних.

До *третього рівня* віднесемо такі дані, у яких доступ до їхніх складових може здійснюватися не лише у такий спосіб, як до структурованих даних, а й із використанням «мінус штрих-операцій». До типів даних третього рівня належать дані, які описуються предикатними функціями та теоретико-множинними операціями. Потрібно зауважити, що поняття умовного виразу в Адресному програмуванні не збігається з поняттям предикатної функції, оскільки у предикатах як аргументи й тіло функції можуть міститися невизначені (вільні) адреси. Типи даних третього рівня називатимемо *декларативними типами даних*.

Для розуміння типів даних третього рівня дамо чітке визначення «мінус штрих-операції» і наведемо приклади її використання.

«Мінус штрих-операція»

«Мінус штрих-операція» є оберненою (зворотною) до «штрих-операції» [1; 2]. Для лінійного списку, зображеного на рис. 7, результат цієї операції однозначно визначений для всіх вузлів списку, окрім першого (голови). Для голови списку результату не існує (операція не визначена). Може так статися, що на одну адресу вказують декілька значень інших адрес (вказують декілька вказівників). У таких випадках результат оберненої операції буде багатозначним. У математиці не прийнято розглядати багатозначні функції. Задля уникнення багатозначності результату «мінус штрих-операції» її результатом вважають сукупність відповідних адрес.

Нехай **A** – певна адреса. Тоді $^{-}{}'\mathbf{A}$ є такою сукупністю усіх адрес, значення яких збігаються з адресою **A**. Формально можна визначити так:

$$^{-}{}'\mathbf{A} = \{\mathbf{B} \mid {}'\mathbf{B}=\mathbf{A}\}$$

На рис. 8 подано лінійний однозв'язний список із застосуванням «мінус штрих-операції», який було розглянуто на рис. 7.

Рис. 8. Однозв'язний список
в Адресному програмуванні із властивостями
(можливостями) двозв'язного списку

Застосування «мінус штрих-операції» дає змогу від адреси будь-якого вузла, окрім першого, отримати адресу попереднього вузла та перейти до нього, не зважаючи на те, що у вузлі немає поля (вказівника) на попередній вузол. У сучасних імперативних мовах це відповідає двозв'язному списку: потрібно до кожного вузла додати поле «**Prev**». Проведемо аналогію з реальним світом. Розглянемо будь-яку чергу людей за якимись товарами. Для визначення повної інформації про послідовність людей у черзі достатньо кожній людині спитати крайнього та запам'ятати його. Ця інформація є вичерпною щодо слідування людей у черзі. Припустимо, що людина побажала дізнатися, хто стоїть у черзі за нею. Для цього ця людина може запитати у всіх (чи у кожного): «Хто зайняв чергу за мною?». Відгукнеться відповідна людина. Засобами імперативного програмування можна скласти відповідний алгоритм, який буде послідовно переглядати інформацію у вузлах списку та знайде потрібний вузол. Використання «мінус штрих-операції» містить у собі опис того, що має бути в результаті, а опису самого алгоритму – немає.

Розглянемо представлення дерева за допомогою «мінус штрих-операції». Аналогічно, як у випадку з чергою, для визначення вичерпної інформації щодо дерева достатньо вказати (запам'ятати) батька для кожного вузла, окрім кореня дерева. Цієї інформації достатньо для визначення сукупності синів кожного вузла. У декларативному програмуванні, як і в логічному та РБД, є можливість визначати дерева саме таким способом.

Розглянемо приклад представлення дерева у РБД. Із погляду РБД дерево являє собою рекурсивне відношення один до багатьох, задане на декартовому добутку вузлів. До атрибутів таблиці «Вузли дерева» визначається атрибут «Батько», який є зовнішнім ключем до рядка таблиці «Вузли дерева». За допомогою використання конструкції Join мови SQL легко написати запит, який за первинним ключем вузла визначить сукупність рядків, які є синами. «Мінус штрих-операція», застосована до адреси вузла дерева, як результат повертає сукупність адрес синів цього вузла.

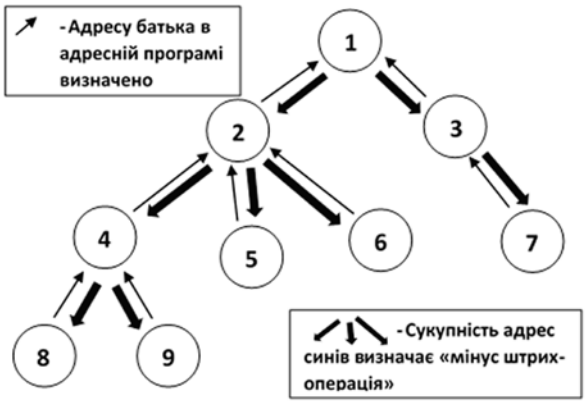

Рис. 9. «Мінус штрих-операція» визначає синів вузлів за браком цієї інформації у вузлах дерева

На рис. 9 наведено приклад представлення дерева з довільною кількістю синів. При цьому адреса будь-якої вершини дерева являє собою тип даних: «це дерево з даними у вузлах». Якщо накласти обмеження на подібність даних у вузлах дерева в Адресній мові програмування, будуть аналогічні таблиці в РБД.

Отже, будь-яку реляційну та фізичну модель РБД, шляхом накладання певних обмежень, може бути представлено як відповідний їй тип даних Адресного програмування.

Висновки

1. Типи даних Адресної мови мають неявну типізацію і допускають довільні типи даних та довільні їх групування між собою, між групами даних. Як складові згрупованих даних можна використовувати функції, подібно до методів у ООП.

2. Адресна мова програмування є багатоконцептуальною та поєднує у собі концепції імперативного та декларативного програмування. Основою декларативної концепції Адресного програмування (1955 р.) є «мінус штрих-операція».

Список літератури

1. Гнеденко Б. В. Элементы программирования / Б. В. Гнеденко, В. С. Королюк, Е. Л. Ющенко. – Москва : ГИФМЛ, 1961. – 348 с.
2. Ющенко Е. Л. Адресное программирование / Е. Л. Ющенко. – Киев : Гос. издательство технической литературы, 1963. – 287 с.
3. Ющенко Е. Л. Адресное программирование и особенности решения задач на машине «УРАЛ» / Е. Л. Ющенко ; под ред. М. М. Бушко-Жук. – Киев : Киев. высш. инженерное радиотехническое училище войск противовоздуш. обороны страны, 1960. – 192 с.
4. Ющенко Ю. О. Багатовимірне впорядкування та його використання для вдосконалення інтерфейсу користувачів інформаційних систем / Ю. О. Ющенко // Наукові записки НаУКМА. – 2018. – Т. 1. Комп'ютерні науки. – С. 10–13.
5. Ющенко Ю. О. Використання багатовимірного впорядкування для наочного та зручного доступу до інформації /

Ю. О. Ющенко // Матеріали XV Міжнародної науково-практичної конф. «Інформаційні технології в економіці, менеджменті і бізнесі. Проблеми науки, практики та освіти» (Київ, 25–26 листопада 2010 р.). – Київ : Вид-во Європ. ун-ту, 2010. – С. 114–115.

6. Ющенко Ю. О. Вступ до логічного програмування : навч. посібник / Ю. О. Ющенко. – Київ : Вид-во Європ. ун-ту, 2006. – 116 с. : іл.
7. Ющенко Ю. О. Засоби керування виконанням логічних програм / Ю. О. Ющенко // Збірник наукових праць МСУ. – 2005. – С. 106–171.
8. Videla Alvaro. Kateryna L. Yushchenko – Inventor of Pointers [Electronic resource] / Alvaro Videla. – Mode of access: https://medium.com/a-computer-of-ones-own/kateryna-l-yushchenko-inventor-of-pointers-6f2796fa1798?fbclid=IwAR3fcqmC0COfy5EqyIHBrlQhCpno5MUFZjCUQ-SM-v-xhD0g3xbj_P2SRCM.

References

Hnedenko, B. V., Koroliuk, V. S., & Yushchenko, E. L. (1961). *Elementy prohrammyrovanyia*. Moskva: HYFML [in Russian].

Videla, Alvaro. (2018). Kateryna L. Yushchenko – Inventor of Pointers. Retrieved from https://medium.com/a-computer-of-ones-own/kateryna-l-yushchenko-inventor-of-pointers-6f2796fa1798?fbclid=IwAR3fcqmC0COfy5EqyIHBrlQhCpno5MUFZjCUQ-SM-v-xhD0g3xbj_P2SRCM.

Yuschenko, Yu. O. (2005). Zasoby keruvannia vykonanniam lohichnykh prohram. *Zbirnyk naukovykh prats MSU*, 106–171.

Yuschenko, Yu. O. (2006). *Vstup do lohichnoho prohramuvannia*. Kyiv: Vyd-vo Yevrop. un-tu [in Ukrainian].

Yushchenko, E. L. (1960). Adresnoe prohrammyrovanye y osobennosty reshenyia zadach na mashyne "URAL". Kiev: Kyev. vyssh. ynzhenernoe radyotekhnycheskoe uchylyshche voisk protyvovozdush. oborony strany [in Russian].

Yushchenko, E. L. (1963). *Adresnoe prohrammyrovanye*. Kyev: Hos. yzdatelstvo tekhnycheskoi lyteratury [in Russian].

Yushchenko, Yu. (2010). Vykorystannia bahatovymirnoho vporiadkuvannia dlia naochnoho ta zruchnoho dostupu do informatsii. In *Materialy XV Mizhnarodnoi naukovo-praktychnoi konferentsii "Informatsiini tekhnolohii v ekonomitsi, menedzhmenti i biznesi. Problemy nauky, praktyky ta osvity"* (pp. 114–115). Kyiv: Vydavnytstvo Yevropeiskoho universytetu [in Ukrainian].

Yushchenko, Yu. (2018). Bahatovymirne vporiadkuvannia ta yoho vykorystannia dlia vdoskonalennia interfeisu korystuvachiv informatsiinykh system. *Naukovi zapysky NaUKMA*, *1*, 10–13 [in Ukrainian].

Yu. Yuschenko

"MINUS STROKE-OPERATION" AND ITS DECLARATIVE ASPECTS

The article considers the capacities of "Minus stroke-operation", which was introduced as the inverse of "stroke-operation" (dereferencing a pointer).

Addressing higher ranks Address Programming is based on "stroke-operation" and, in contrast to the dereference (or indirection) operator, allows you to specify the rank of addressing, which can be both an integer value or an expression of integer type. The paper gives an example of using a "stroke-operation" to obtain the address of an arbitrary element of a linear single-linked list by its sequence number. The dereference operator of modern programming languages does not allow to make it and demands implementation with use of a cycle.

"Minus stroke operation" operation is not used in imperative high-level programming languages, because it belongs to the declarative paradigm of programming.

Address programming language has the dynamic types of data, which are based on "minus stroke-operation".

This article provides some specific examples of the use of "minus stroke-operation" in the Address Programming language and compares the capabilities of this operation with some modern declarative programming tools. From the above examples it follows the universal power of "minus stroke-operation" as a tool of declarative programming.

The paper presents an example of tree representation using "minus stroke-operation". In Address Programming, you can represent trees by defining only the vertex parents. The result of applying to the address of the node "minus the stroke-operation" is a set of sons of this node. An example of a tree representation in the Address Programming Language with parent-only indication is given.

The Address Programming Language is multi-conceptual and combines the concepts of imperative and declarative programming. The basis of the declarative concept of Address Programming (1955) is "minus stroke-operation".

Keywords: stroke operation, minus stroke operation, address sorting, multidimensional address sorting, lists, two-linked lists, trees, abstract data type, data structures, simple data, scalar data, arrays, records, indicators, links, Pointers, data types, implicit typing, implicit data typing, Databases, Relational Databases, SQL, Foreign Key, information systems, logical programming, imperative programming, declarative programming, programming languages.

Матеріал надійшов 11.06.2020