УДК 004.[235+43+655.3]
DOI: 10.18523/2617-3808.2021.4.78-87

Ющенко Ю. О.

ДЕРЕВОПОДІБНІ ФОРМАТИ АДРЕСНОГО ПРОГРАМУВАННЯ

В Адресному програмуванні було введено поняття опосередкованої адресації вищих рангів (Pointers), яка дає змогу довільним чином з'єднувати комірки оперативної пам'яті комп'ютера. Засадами цього з'єднання є стандартне слідування адрес комірок в оперативній пам'яті та адресне слідування, яке визначається опосередкованою адресацією. Використання двох відношень дає можливість визначати довільне об'єднання комірок пам'яті з будь-яким їх змістом. Машинні команди комп'ютера «Київ» надають прямий доступ до елементу «списку» за його порядковим номером. У статті на прикладах однозв'язних «списків» продемонстровано особливості деревоподібних форматів та їх відмінності від абстрактних типів даних. Стаття відкриває нову галузь теоретичних досліджень, метою яких є аналіз доцільності включення у сучасні мови програмування окремих можливостей Адресного програмування.

Ключові слова: вказівники, Pointers, dereference, dereference operator, indirection, multiply indirection, програмування, історія ІТ, історія програмування, архітектура комп'ютерів, деревоподібні формати, абстрактні типи даних, списки, масиви, Адресна мова програмування, Адресне програмування, Ф-операція, комп'ютер «Київ», ЕОМ «Київ».

1. Вступ: історична довідка

Основою об'єднання даних та інших складових програм, зокрема підпрограм (функцій або методів), є опосередкована адресація вищих рангів, до якої подібні вказівники. Вперше опосередковану адресацію другого рангу (вказівники) було використано ще при програмуванні на комп'ютері МЕОМ (рос. МЭСМ)[1].

Можливість використовувати опосередковану адресацію 2-го рангу на МЭСМ забезпечувалась унікальністю архітектури цього комп'ютера, яка полягала в можливості динамічної модифікації команд завдяки так званій команді складання команд. Зокрема команда складання команд МЭСМ давала змогу використовувати комірки оперативної пам'яті МЭСМ як індексні регістри, що являло собою новітню технологію для тих часів. Можливість динамічної зміни програм МЭСМ підказала ідею можливості створення програмуючих програм (ПП), або трансляторів і компіляторів. У 1954 р. на МЭСМ було реалізовано дослідницьку програмуючу програму (транслятор), яка за формулою (арифметичним виразом) створювала програму її обчислення. Досвід, отриманий під час цієї реалізації, підтвердив потребу та корисність опосередкованої адресації.

У 1954 р. було розпочато розроблення великого асинхронного комп'ютера «Київ» широкого призначення та створення для нього мови програмування з опосередкованою адресацією, яка вплинула на включення групових операцій модернізації адрес до системи команд цього комп'ютера. Слід зазначити, що групові операції модернізації адрес можуть виконувати багатократне застосування «штрих-операції» (Multiple indirection of Pointers).

У роботі подано сучасну інтерпретацію поняття деревоподібних форматів Адресного програмування, яке ґрунтується на відношенні слідування комірок оперативної пам'яті комп'ютера. Опосередкована адресація вищих рангів дає змогу довільним чином з'єднувати комірки оперативної пам'яті комп'ютера. Засадами цього з'єднання є стандартне слідування адрес комірок в оперативній пам'яті та адресне слідування, яке визначає програміст опосередкованою адресацією. Використання двох типів відношень дає можливість ввести довільне об'єднання комірок оперативної пам'яті з довільним змістом: дані, адреси, підпрограми, мітки програми тощо. Змістом зазначених адрес можуть бути елементи цього переліку. Своєю чергою, утворені з'єднання комірок можна з'єднувати між собою. Деревоподібні формати, зокрема, дають змогу об'єднувати дані у довільні складні структури даних, подібні до абстрактних типів даних. Тож деревоподібні формати являють собою більш загальні об'єднання порівняно з абстрактними типами даних, оскільки у своєму складі можуть містити

[1] Надалі використовуватимемо назву «МЭСМ», що стала звичною як власна назва.

підпрограми (їхні адреси) та будь-які інші компоненти. Для роботи програм із деревоподібними форматами вводиться поняття схеми огляду [2]. На одному деревоподібному форматі можна визначати декілька схем огляду. Над комірками, з'єднаними деревоподібними форматами, можна створювати оглядові деревоподібні формати для визначення потрібних схем огляду.

У статті продемонстровано відмінність деревоподібних форматів від абстрактних типів даних. На прикладах показано вплив групових операцій модернізації адрес комп'ютера «Київ» на прискорення виконання програм оброблення складних структур даних. Наведено приклади початкових текстів програм оброблення однозв'язних списків на мові С++ та на Адресній мові.

Оскільки існує розбіжність між деревоподібними форматами та абстрактними типами даних, у подальшому поняття «список» [3] у значенні Адресного програмування писатимемо у лапках.

Опосередкована адресація вищих рангів дала змогу розробити універсальну концепцію об'єднання комірок оперативної пам'яті. Згідно з довільними потребами будь-яких предметних галузей і залежно від потреб задач, які має розв'язувати програма, програмісти мають можливість створювати довільні об'єднання комірок пам'яті комп'ютера.

Унікальною особливістю використання деревоподібних форматів було те, що для прискорення їх оброблення у процесорах комп'ютерів були реалізовані групові операції модернізації адрес, винайдені київськими вченими та вперше реалізовані в системі команд комп'ютера «Київ».

Самі по собі групові операції комп'ютера «Київ» становлять засіб організації циклічних процесів, який є у мовах програмування високого рівня: заголовок циклу та тіло циклу. Система команд комп'ютера «Київ» містила більше засобів та можливостей, ніж перша мова програмування високого рівня, «Планкалькюль» Конрада Цузе (1944 р.), оскільки мала, окрім операції «засилання» (присвоєння), умовних та безумовних переходів, засоби організації циклічних процесів на кшталт операторів циклів імперативних мов програмування високого рівня та засоби опосередкованої адресації (Pointers). Отже, машинну мову процесора комп'ютера «Київ» слід відносити до мови програмування високого рівня. Так з'являється адресний метод програмування, якому згодом, у 1958 р. надається назва «Адресна мова програмування». Спочатку математики писали програми у мнемонічних кодах адресного методу програмування, а математики-обчислювачі переводили ці записи у машинні команди. Також існували програми для комп'ютера «Київ», які відповідали машинним командам із використанням мнемонічних позначок кодів операцій та кодів операндів (адрес) [1, с. 125]. Для комп'ютера «Київ» розробили програмуючу програму (або компілятор) «**Адресна мова ЕОМ "Київ"»** (скорочено – ПП-АК) [1, с. 125–148; 7, с. 144] та ПП-2[4] [1, с. 148–160; 5]). У 1955 році на комп'ютері «Київ» почали писати та налагоджувати службові програми та реалізовувати стандартні бібліотечні підпрограми змінно-спаяної оперативної пам'яті, а з 1956 р. почали активно використовувати для потреб народного господарства.

Адресна мова програмування була стандартною мовою, яку використовували на комп'ютерах радянського виробництва у 50-х і 60-х роках минулого сторіччя: «Київ», «М-20», «Урал», «Мінськ», «Дніпро», БЭСМ-3, БЭСМ-3М, БЭСМ-4 тощо. Більшість із цих комп'ютерів постачалися до соціалістичних країн, а деякі з названих серійно виготовляли у КНР, хоча і з іншими назвами. Загальний строк використання програмістами Адресної мови на всьому постсоціалістичному просторі становив понад 20 років.

Загальновідомо, що за кордоном Pointers (опосередкована адресація 2-го рангу) було винайдено лише у 1964 р. Гарольдом Лоусоном [12]. Цей винахід повторює розробку опосередкованої адресації вищих рангів [1; 2; 7; 11], а дослідження закордонними вченими застосувань вказівників, зокрема абстрактних типів даних, багато в чому повторили дослідження українських учених. Так сталося, що 1965 р. комуністичний тоталітарний режим відмовився від подальшого використання Адресної мови програмування на користь мов FORTRAN та ALGOL-60. Роботи зі створення інтерпретаторів Адресної мови для вже нових комп'ютерів радянського виробництва було припинено. Закордонним ученим, через «залізну» стін часів холодної війни, не було відомо про винаходи київських учених щодо опосередкованої адресації вищих рангів та деревоподібних форматів. Архітектура комп'ютера «Київ», як і інших радянських комп'ютерів,

[2] В Адресній мові програмування використовували терміни «огляд» (рос. обозревание). Ми надалі вживатимемо термін «обхід» (за аналогією до обходу дерев та інших абстрактних типів даних). Термін «схема обходу» відповідає за змістом способу або методу обходу абстрактних типів даних.

[3] У джерелах щодо Адресного програмування термін «список» не використовували.

[4] Вхідна Адресна мова для ПП-2 призначена виключно для комп'ютера «Київ» та використовує особливості системи команд цього комп'ютера.

разом з Адресною мовою програмування значно випереджали закордонні інформаційні технології. Українські вчені та історики науки у своїх публікаціях не зазначають про ці великі здобутки, хоча в усьому світі визнано надзвичайну значущість опосередкованої адресації вищих рангів (Pointers) для інформаційних технологій.

Опосередкована адресація (Pointers) дає змогу об'єднувати довільним чином між собою комірки оперативної пам'яті та являє собою невід'ємну складову всіх сучасних технологій програмування. У логічних, функційних мовах програмування та в деяких імперативних мовах (наприклад, JavaScript, Python тощо) немає типу даних «вказівник». Однак реалізації цих мов, зокрема реалізація списків у цих мовах, неявно використовують вказівники. Об'єктно-орієнтоване програмування також базується на опосередкованій адресації, яка дає змогу поєднувати дані з процедурами (методами) їх оброблення.

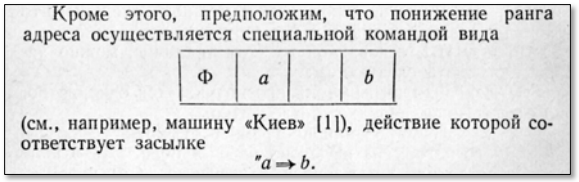
Реалізація реляційних та інших типів баз даних також базується на використанні опосередкованої адресації. В [14] досліджено окремі аспекти декларативності штрих-операції та зроблено висновок, що Адресна мова програмування має вичерпні декларативні можливості. Завдяки універсальним можливостям з'єднання даних у деревоподібні формати та декларативним можливостям наприкінці 50-х років минулого сторіччя на комп'ютері «Київ» було розроблено першу в світі табличну базу даних реляційного типу – «Автодиректор» [4].

У статті проаналізовано розбіжності між класичною концепцією абстрактних типів даних і концепцією деревоподібних форматів, зроблено висновок щодо існування окремих нюансів у деревоподібних форматах, які не мають аналогів у сучасних технологіях імперативного програмування.

2. Апаратна реалізація операцій із Pointers

Потреби мови програмування, з урахуванням вимог зручності для програмістів, зумовили включення виконання «штрих-операції» (розіменування вказівників) до системи команд процесора комп'ютера «Київ». Зокрема, Ф-операція із системи команд комп'ютера «Київ» давала змогу зменшувати ранг адреси [1, с. 14; 7, с. 146], тобто виконувати розіменування вказівника або двічі застосовувати «штрих-операцію» Адресної мови програмування (див. рис. 1 [5]).

[5] Адресній формулі **''a ⇒ b** у мові **C++** відповідає розіменування вказівника: **b = *a**.


Кроме этого, предположим, что понижение ранга адреса осуществляется специальной командой вида

Ф	a		b

(см., например, машину «Киев» [1]), действие которой соответствует засылке

"a ⇒ b.

Рис. 1. Фрагмент із [8, с. 146] (міститься посилання на [2, с. 14])

Для зручності програмування дій на комп'ютері «Київ», які мають повторюватися багатократно (циклічно), до операцій процесора включають можливості визначення циклів. Ф-операція «*дає змогу кодувати в постійній пам'яті циклічні процеси довільної глибини*» [1, с. 14]. Для цього використовують так звані групові операції модернізації адрес, щоб послідовно оглядати та виконувати дії не лише з елементами масивів, а й з елементами «списків». Також ці групові операції допомагають отримувати прямий доступ до довільного елементу «списку» за його порядковим номером – **^{i}List**, на кшталт прямого доступу до елементу масиву за його індексом – **M[i]**.

Можливості визначати циклічні дії груповими операціями модернізації адрес машинної мови комп'ютера «Київ» відповідають «заголовкам циклів» в імперативних мовах програмування та навіть мають більше можливостей, оскільки можуть визначати повторення дій і для змінних, які «пробігають» «список». Іншими словами, групові операції модернізації адрес можна використовувати не лише для огляду (або обходу) послідовно розташованих у комірках пам'яті комп'ютера елементів масивів, а й для послідовного переходу від одного елементу «списку» до наступного.

3. Відношення слідування комірок оперативної пам'яті комп'ютера

Одним із базових понять Адресного програмування є поняття відношення слідування та поняття «наступної» комірки оперативної пам'яті. Розрізняють два типи цих відношень слідування.

Перший тип – це «природне» слідування комірок оперативної пам'яті комп'ютера, яке зумовлено її лінійністю. Для кожної (окрім останньої) комірки пам'яті однозначно визначається наступна комірка, адреса якої на 1 більше за поточну. В Адресному програмуванні таке відношення слідування називають **стандартним відношенням слідування (або стандартним слідуванням адрес)**.

Стандартному слідуванню адрес відповідає слідування адрес елементів масивів (будь-якої вимірності), оскільки елементи масивів розташовуються у пам'яті комп'ютера послідовно один за іншим.

В Адресній мові допускаються арифметичні дії віднімання адрес **A - B**. Результатом цієї операції може бути **0**, додатне або від'ємне ціле число.

В Адресній мові використовують арифметичні операції з адресами, результатом яких є адреса. Такі арифметичні операції позначають у кружечку, наприклад: ⊕, ⊗ тощо.

Залежно від типу елементів масивів для збереження кожного елемента масиву виділяється відповідне ціле число комірок. Якщо елементи масиву зберігаються у трьох комірках пам'яті, то адреса комірки кожного наступного елементу на **3** більше, ніж адреса попереднього. Якщо **A** – адреса такого елементу масиву, то **A+3** буде адресою наступного за **A** елемента. При цьому вважають, що **A+3 = A ⊕ 1**. Тобто «⊕ **1**» – визначає наступний елемент масиву незалежно від кількості комірок пам'яті, потрібних для збереження значення, та реалізує стандартне слідування адрес.

Кожній схемі обходу адрес однозначно відповідає схема обходу значень, які в цих адресах зберігаються. Схему обходу значень згідно зі стандартним слідуванням адрес можна визначати у порядку збільшення адрес або у зворотному порядку – у порядку зменшення. Схеми обходу можна визначати й іншим способом, наприклад за допомогою арифметичних формул.

Аналогічно визначають стандартний тип слідування для адрес складових частин простих структур даних [6]. Значення складових частин таких структур можуть мати різні типи, які в оперативній пам'яті займають різну кількість комірок. Звернення до складових частин простих структур даних здійснюється за порядковим номером складової частини (в імперативних мовах – за іменем структури та іменем складової частини) або за її адресою.

Доступ у програмах імперативних мов програмування до значень простих скалярних змінних, до елементів масивів та до складових простих структур даних здійснюється **адресацією першого рангу, або прямою адресацією**.

[6] Під простою структурою розумітимемо об'єднання даних різного типу, зокрема підструктури, масиви даних та масиви підструктур. Елементарні (прості, скалярні) компоненти, або складові частини, структур можуть мати різний тип даних. Звернення до складових частин простих структур даних здійснюється за іменем структури та іменем підструктури. До елементів масивів структур та масивів підструктур звернення здійснюється за індексом елементу.

До другого типу відношення слідування належить слідування комірок пам'яті комп'ютера, яке задається адресами, у яких зберігаються наступні адреси. Таких відношень слідування, або **адресного слідування,** програміст може створювати стільки, скільки йому потрібно.

Якщо комірка пам'яті комп'ютера як своє значення містить адресу іншої комірки, в якій зберігається значення, то таку адресацію називають **опосередкованою (непрямою) адресацією другого рангу**. Використання будь-якого вказівника передбачає використання адресації другого рангу.

Адресація вищих рангів дає змогу утворювати ланцюжки адрес довільної довжини: «адреса на адресу іншої адреси, яка теж вказує знов на адресу і так далі». Довжина ланцюжка визначає ранг адреси.

Записи констант у програмі являють собою **адресацію 0-го рангу.**

Як уже зазначено, у програмі можна створювати таких ланцюжків стільки, скільки потрібно. До особливих і виняткових ланцюжків належать:

а) порожні ланцюжки: адреси, за якими міститься адреса «**0**» («**Null**») – аналог порожнього списку;
б) ланцюжки з одного елементу (адреси простих скалярних змінних, елементи масивів та складові простих структур даних), тобто пряма адресація або адресація 1-го рангу;
в) циклічні ланцюжки, якщо адреса вказує на одну з адрес, яка передувала.

Зауважимо, що різні ланцюжки можуть мати спільні частини – перетинатися. Так само, як і однозв'язні списки, у класичному розумінні, можуть бути циклічними або мати наприкінці спільні частини.

Кожний такий ланцюжок назвемо «списком» у розумінні Адресної мови програмування.

4. Область доступності

Важливим поняттям Адресного програмування є область доступності. Вводяться області доступності для програми, фрагмента програми, адресної функції, змінної.

Якщо змінна є простою скалярною, то область її доступності – адреса (ім'я цієї змінної). Для елементу масиву, як першого, так і кожного іншого, областю доступності є адреси всіх елементів масиву. Аналогічно, областю доступності для простої структури даних та для довільної її складової частини є сукупність всіх адрес структури.

Кожна адресна функція зберігається за певною адресою оперативної пам'яті. Кожен рядок

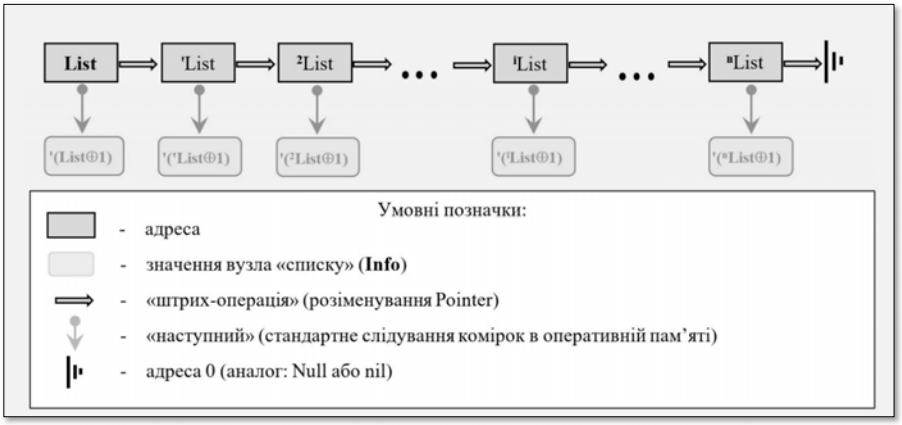


Рис. 2. «Список» – простий деревоподібний формат Адресного програмування (аналог списку в імперативних мовах програмування)

адресної програми має адресу, з якої цей рядок починає зберігатися в оперативній пам'яті комп'ютера. Область доступності для адресної функції та адресної формули – це об'єднання всіх областей доступності їхніх складових частин. Для довільного фрагмента програми область доступності – це об'єднання областей доступності адресних формул, які входять у цей фрагмент, з адресами комірок, де розташовані адресні формули та їхні складові. Мітка адресної формули (рядка адресної програми) є адресою першої комірки, з якої розташована ця адресна формула.

Універсальність області доступності при створенні програмістом зв'язків між даними дала змогу ще у 50-х роках минулого сторіччя ввести у програмування термін «адресне сортування (впорядкування)» [6–9]. Під адресним сортуванням розуміють створення такої структури даних, яка містить вичерпну інформацію про порядок слідування елементів, які впорядковують не шляхом зміни місця їх розташування, а шляхом «сортування» адрес, які вказують на ці елементи.

Адресне сортування дало змогу визначити та використовувати багатовимірне адресне сортування, коли набір елементів впорядковується одночасно за багатьма різними атрибутами. За багатовимірного адресного сортування елементи залишаються на місцях свого розташування, а під час сортування створюється структура даних (деревоподібний формат), у якій, після завершення сортування, буде міститися вичерпна інформація про впорядкування елементів окремо за кожним із атрибутів.

Багатовимірне адресне сортування використовують для вирішення широкого класу задач, зокрема щодо пошукових систем, задач класифікації тощо. Адресне сортування застосовують у реляційних базах даних, коли індексуються таблиці за зростанням одного значення одного з полів [6–9].

Поняття схеми обходу області доступності відповідає обходу деревоподібного формату.

5. Деревоподібні формати

Сам по собі ланцюжок адрес («список») являє собою просту форму деревоподібного формату. Більш складні деревоподібні формати утворюються наступними, сусідніми адресами, які входять до складу ланцюжка. На рис. 2 показано приклад «списку» довжиною **n+1** з інформацією у його вузлах. Комірка з адресою **List** містить значення наступної комірки. А комірка з адресою **List** ⊕ **1**, тобто наступна згідно зі стандартним слідуванням адрес, містить значення першого вузла списку.

На зміст адрес ^{i}**List**⊕**1** (**0** ≥ **i** ≥ **n**), які входять до складу деревоподібних форматів, жодних обмежень не накладається. У наведеному на рис. 2 прикладі за цими адресами містяться значення вузлів «списку». У загальному випадку цими значеннями можуть бути й адреси.

Деревоподібному формату відповідає спрямований граф, у якого вершинами графу є адреси, а дуги мають два типи: стандартного та адресного слідування. Дуги стандартного слідування ведуть від адреси одного елементу масиву до адреси наступного елементу та від однієї адреси складової простої структури до адреси її наступної складової.

6. Окремі синтаксичні конструкції Адресної мови

«Адресна мова досить близька до звичайної мови математичних формул» [7, с. 10]. До алфа-

віту Адресної мови входять заголовні і прописні літери латинського, кириличного, грецького алфавітів, цифри та порожня буква – проміжок (відступ або пробіл). Допускається надрядковий і підрядковий записи літер та цифр. До алфавіту включено знаки всіх математичних операцій, зокрема арифметичні, булеві та операції над множинами, наприклад: **+, –, x, *:*, *Y*, ln, sin** і т. д. [7, с. 10]. Для опису алгоритмів до алфавіту Адресної мови додано ряд спеціальних символів. Окремі з цих символів мають спеціалізований сенс.

Адресна програма складається із рядків. Кожний рядок містить запис однієї чи декількох алгоритмічних дій. Запис кожної дії називають адресною формулою [7].

Формули в одному рядку відокремлюють комами – «**,**» або крапкою з комою – «**;**». Крапка з комою визначає можливість виконання адресних формул у довільному порядку чи одночасно [8] [7, с. 23].

В Адресній мові використовують поняття «адресне відображення» (або «адресне відношення»). Адресне відображення являє собою бінарне відношення, яке визначається на двох множинах: **A** та **B**: **Ar: A → B** [7, с. 10, 11].

Адресне відношення являє собою всюди визначене функціональне відношення, тобто для кожного елемента множини **A** обов'язково існує один (у жодному разі не більше) елемент множини **B**, такий, що **(a, b)** ∈ **Ar**. На практиці зручно вводити спеціальні значення, які трактуються як «значення не має». Адреса «нульової» комірки пам'яті «**0**» або «**Null**» є прикладом такого значення.

Позначають адресне відображення символом «штрих» – «**ʻ**»: **ʻa = b**. Цей запис можна читати так:

а) «штрих **a** дорівнює **b**»;
б) «пара **(a, b)** належить адресному відношенню»; або
в) за адресою **a** зберігається значення **b**.

Будемо використовувати і такі словосполучення: «значенням за адресою **a** є **b**», «значення **b** зберігається за адресою **a**», «змістом адреси **a** є значення **b**», «**a** містить **b**» та «**b** є вмістом **a**». «Всі ці висловлювання є рівнозначними» [7, с. 11].

Відношення (відображення), яке обернене до адресного відображення, може бути неоднозначним (нефункціональним), оскільки за різними адресами можуть зберігатися однакові значення.

Визначене адресне відображення називають **«штрих-функцією»**. У загальному випадку адресне відображення визначають на довільних абстрактних множинах **A** та **B,** і їх необов'язково прив'язувати до комп'ютерів і мов програмування. Під **«штрих-операцією»** розуміють визначення значення **«штрих-функції»** для конкретного її аргументу.

В Адресній мові передбачено правила запису для обох множин **A** та **B**. Для адрес (імена змінних являють собою аналог) можна використовувати довільні послідовності літер із заголовних, прописних літер латинського, кириличного, грецького алфавітів та цифр, зокрема їх надрядковий та підрядковий записи. Ці послідовності не мають містити пробіли, спеціальні знаки і символи математичних функцій. До множини адрес не належить «**0**». Програмісти розуміють під «**0**» адресу першої комірки оперативної пам'яті, у якій нічого не міститься та в яку не можна нічого заслати. В сучасних мовах програмування використовують, окрім «**0**», записи «**Null**» або «**nil**».

Правило визначення елементів множини **B** визначається аналогічно, як правило запису констант у сучасних мовах програмування. Як зазначено вище, адресне відображення є всюди визначеним, тобто за кожною адресою зберігається значення. Якщо **a** не належить множині адрес, то **ʻa** – не має сенсу. У загальному випадку, множини **A** та **B** можуть перетинатися. У цьому разі виникає опосередкована адресація вищих рангів (або вказівники).

Однократне застосування «штрих-операції» являє собою пряму адресацію. У сучасних мовах програмування це відповідає зверненню до значень простих, скалярних змінних. До прямої адресації належить звернення до значень елементів масивів та складових простих структур. Багатократне застосування (понад один раз) «штрих-операції» – це опосередкована адресація вищих рангів.

Багатократне застосування «штрих-операції» в Адресній мові позначають декількома штрихами, наприклад: **ʻʻa**, **ʻʻʻʻa**. Можна використовувати й таку форму запису: ^{2}**a,** ^{4}**a,** або таку: ^{i}**a**, $^{'i}$**a,** де **i** може бути довільна адресна функція, яка набуває ціле значення. У записах типу $^{-4}$**a** операція матиме назву «мінус штрих-операція» [10], яка є оберненою до «штрих-операції» і може бути неоднозначною. Результатом «мінус штрих-операції» є сукупність адрес, які як значення містять її аргумент.

В Адресній мові визначають формулу засилання [9], яку записують так: **a** ⇒ **b**. Тут **a** та **b** –

[7] Поняття «оператор» в імперативних мовах програмування збігається із поняттям «адресна формула».

[8] Можливість паралельного (одночасного) виконання адресної програми було використано для визначення розподілених процесів асинхронного комп'ютера «Київ».

[9] Оператор присвоєння в імперативних мовах подібний до формули засилання.

адресні функції [10]. Значенням адресної функції **a** може бути довільний елемент множин **A** та **B**, а значенням функції **b** має обов'язково бути елемент множини **B**. Формула засилання змінює адресне відображення у такий спосіб: елемент адресного відношення $(\mathbf{b}_0, \mathbf{c}_0)$ замінюється на елемент $(\mathbf{b}_0, \mathbf{a}_0)$, де $\mathbf{b}_0$ та $\mathbf{a}_0$ – значення адресних функцій **b** та **a**, а $\mathbf{c}_0$ – колишнє значення адресного відображення для **b**.

Позначку «⇒» Конрад Цузе використав у мові програмування «Планкалькюль» для визначення зміни значень за адресами. Така позначка природно сприймається, оскільки спочатку підраховується значення **b**, а потім воно засилається в **a**. З погляду математиків «=» для цього не дуже підходить, адже вони цю позначку використовують для визначення рівності.

Наближеною до формули засилання «» є формула обміну: **a** ⇔ **b** [7, с. 22]. Виконання цієї формули відбувається так:

1) визначають значення адресних функцій, які мають бути адресами;
2) відбувається обмін значень, які містяться у цих адресах.

Або виконується дві формули засилання – **'a** ⇒ **b** та **'b** ⇒ **a**, однак обидва засилання виконуються після визначення **'a** та **'b**. Формула обміну **a** ⇔ **b** рівносильна послідовному виконанню чотирьох формул засилання: **a** ⇒ **q, 'b** ⇒ **p, 'a** ⇒ **b, 'p** ⇒ **'q**. Тут адреси **p** та **q** – тимчасові (допоміжні). Може скластися враження, що достатньо трьох таких формул засилань: **'a** ⇒ **p, 'b** ⇒ **a, 'p** ⇒ **b**. Але це не так, оскільки **b** може бути залежним від **'a**. У цьому випадку друге засилання зміною значення **a** спотворить адресу **b**, оскільки вона залежить від **a**.

В імперативні мови програмування можна ввести лише спрощений варіант обміну значень змінних.

В Адресній мові є формули умовних та безумовних переходів. Умовні переходи використовують так звані розпізнавачі, яким відповідають умови в сучасних мовах програмування. За допомогою безумовних переходів в Адресній мові можна визначати дії, які мають повторюватися циклічно.

Для впорядкування елементів множин вводиться операція слідування – ***C***. «У кожному конкретному випадку операція слідування має описуватись алгоритмічно» [7, с. 12]. Для стандартного та адресного слідування адрес ця операція апаратно реалізована Ф-операцією. Як операцію слідування при обході деревоподібного формату може бути використано довільний запрограмований алгоритм. Окрім цього, програмісти мають можливість над елементами довільних множин визначати нові відношення слідування адрес, які можна динамічно змінювати.

Наведемо приклад використання декількох різних операцій слідування на елементах однієї множини. Припустимо, що при роботі з елементами множини необхідно користуватися результатами сортування її елементів за різними критеріями. Для цього можна визначити декілька різних схем обходу елементів цієї множини. Кожна схема обходу буде визначати порядок слідування елементів згідно з одним із критеріїв. Саму схему обходу елементів множини можна визначити «списком» (ланцюжками адрес), тобто визначити окреме для кожного критерію відповідне адресне слідування.

Слід окремо зауважити, що стандартне та адресне слідування у концепції Адресного програмування подібні та реалізовані апаратно однією операцією – Ф-операцією. Ця особливість Адресного програмування дає можливість визначати схеми обходу елементів «списків» так само, як і схеми обходу елементів масивів. Запис формули циклювання («заголовку циклу») за елементами списку подібний до запису формули циклювання за елементами масивів та не складніший за нього, на відміну від запису алгоритму циклювання за елементами списку у сучасних мовах програмування. Продемонструємо це на прикладах у наступному пункті.

7. Особливості «списків» в Адресному програмуванні

Приклад 1

Вказівники, які утворюють ланцюжок («список»), можуть бути не скалярними, а входити до складу масивів або структур. Саме це дає змогу «порожні», на перший погляд, ланцюжки навантажувати додатковою інформацією. Якщо вказівники ланцюжка входять до складу структури з двох елементів (адреса, інформація), то такі «списки» стають подібними до класичних списків. Тобто стандартне слідування до адреси з ланцюжка визначає значення інформації (**Info**) у відповідного вузла «списку». На рис. 2 зображено «список» із **n+1** елементом.

Розбіжність поняття «списку» Адресного програмування із класичним поняттям списку полягає у тому, що доступ до **i**-го елементу «списку» можна отримати **i**-кратним застосуванням «штрих-операції» (Multiply indirection), яка

[10] Поняття адресної функції є очевидним, тому не наводимо його точне визначення.

Таблиця 1

Приклади фрагментів програм із циклом за елементами масиву та елементами списку

	C++	Адресна мова	Пояснення на мові, подібній до C++

Цикл за елементами масиву	**for** int **i** (**0**; **n**, **++i**)
	{
	F(M[**i**]);
	}
		Ц{ **0**, (∅⊕**1**, **n**) ⇒ π}
		П F{ }
			-

Цикл за елементами списку	List *Current =Head;
	while (Current -> Next != Null)
	{
	F(Current->info);
	Current = Current -> Next;
	}
		Ц{ List, (‘’∅) ⇒ π}
		П F{π ⊕1}
			// подібного в сучасних мовах немає:
			for *Pi (List; Null, *List)
			{
			F(Pi);
			}

виконується груповою операцією модернізації адрес. Окрім цього, для проходження по елементах списку в сучасних імперативних мовах програмування необхідно використання змінної (**Current**), а в концепції Адресного програмування групова операція модернізації адрес, замість цієї змінної, використовує спеціальний регістр модернізації адрес, що зменшує кількість операцій звернень до оперативної пам’яті. Тож апаратна реалізація багатократного застосування «штрих-операції» у системі команд комп’ютера з використанням регістрів процесора замість оперативної пам’яті прискорює процес обходу «списку».

Крім того, в Адресній мові програмування існують заголовки циклів не тільки зі змінними циклу арифметичного типу, а й зі змінними типу «вказівник». Можна визначати перший елемент, із яким має виконуватись перший раз тіло циклу. Замість кроку (**Step**) можна вказати адресне слідування («штрих-операцію). При цьому, якщо не вказувати додаткові умови, цикл буде виконуватись для всіх елементів «списку». В сучасних імперативних мовах необхідно введення змінної **Current** запису **Until** або **While** з перевіркою відповідної умови. В табл. 1 наведено приклад фрагментів із початкових текстів на мові **C++** та на Адресній мові, в яких циклічно виконуються дії **F** із кожним елементом масиву та списку.

Літери Ц та П – це синтаксичні позначки формули циклювання та формули входження (виклику) підпрограм відповідно.

Літера ***π*** являє собою аналог змінної циклу **i** та змінної **Current**, а її значення зберігається у регістрі. Запису ***Ц*{ List, (‘’∅) ⇒ }**[11] відповідають апаратно реалізовані групові операції модернізації адрес, до складу якої входить Ф-операція. Апаратна реалізація та використання регістру прискорюють виконання наведених у таблиці циклів.

Наведеному прикладу оброблення елементів списку відповідає функція **map** (наприклад, у функційній мові Haskell). В Адресному програмуванні цю функцію можна визначити так:

map … ∅ ⇒ F, ∅ ⇒ List

Ц{ List, (‘’) ⇒ π}

П F{⊕ 1}

При цьому функція F замінює значення, яке було за адресою вхідного параметра, результатом свого виконання.

Якщо у вузлах «списку», окрім інформації, у наступній комірці за значенням зберігається функція, яку потрібно застосувати до цього значення, то функцію **fmap** можна реалізувати так:

fmap … ∅ ⇒ F, ∅ ⇒ List

Ц{List, (‘’) ⇒ π}

П (π ⊕ 2){π ⊕ 1}

Приклад 2

Припустимо, що для розв’язку задачі потрібно маніпулювати з елементами впорядкованої множини. При цьому через специфіку задачі потрібно часто змінювати визначений порядок. Розглянемо приклад представлення впорядкованої множини «списком» (рис. 3). Значення вузлів списку, на відміну від «списку», представленого на рис. 2, зберігатимемо з використанням адресації 2-го рангу.

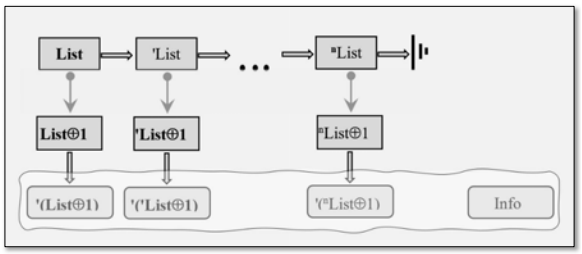


Рис. 3. Приклад «списку» зі збереженням інформації вузлів з адресації 2-го рангу

У табл. 2 наведено фрагмент програми на Адресній мові, який міняє місцями два елементи (інформацію у вузлах «списку»): **3** із **5** та **i** з **j**. На мові Пролог це можна лаконічно записати лише за фіксованої кількості елементів списку (мно-

[11] В асемблерах сучасних комп’ютерів такої конструкції немає.

Таблиця 2

Фрагмент програми заміни місцями елементів 5 і 8 «списку»

Адресна мова	Пролог	Імперативні мови
^{3}List⊕1 ⇔ ^{5}List⊕1	p([_, _, A3, _, A5, _, _], [_, _, A5, _, A3, _, _]).	Можна запрограмувати з використанням двох циклів
^{i}List⊕1 ⇔ ^{j}List⊕1	Можна записати з використанням двох рекурсій та лічильника	

жини). У сучасних імперативних мовах для зміни місцями двох елементів списку необхідно використати два цикли.

Стисліcть, простота й лаконічність запису в Адресній мові досягається тим, що до елементів списків можна здійснювати прямий доступ за їхніми порядковими номерами, тобто так само, як здійснюється прямий доступ до елементів масивів за їхніми порядковими номерами.

8. Висновки

Між поняттями деревоподібних форматів Адресного програмування та абстрактними типами даних є багато спільного. Однак є відмінність, яка впливає на лаконічність запису та простоту сприйняття алгоритмів із використанням деревоподібних форматів порівняно з абстрактними типами даних. Це продемонстровано на прикладах «списків». Апаратно реалізовані групові операції модернізації адрес прискорюють оброблення складних структур. Деревоподібні формати – більш загальне поняття порівняно з абстрактними типами даних, оскільки як компоненти допускають не лише дані, а й, зокрема, підпрограми. Деревоподібні формати являють собою цікаві, з теоретичного погляду, засоби об'єднання змісту оперативної пам'яті; дають змогу з'єднувати дані з програмами їх оброблення, розміщувати (обгортати) їх у контейнери.

Багато прийомів опосередкованої адресації вищих рангів Адресного програмування мають відображення у сучасних сферах застосування вказівників. Однак існують окремі нюанси використання опосередкованої адресації вищих рангів, яких немає у сучасних засобах та технологіях програмування. Відмінності деревоподібних форматів від абстрактних типів даних відкривають нову галузь теоретичних досліджень, метою якої є аналіз доцільності залучення окремих можливостей Адресного програмування до сучасних мов і технологій програмування.

Список літератури

1. Глушков В. М. Вычислительная машина «Киев». Математическое описание / В. М. Глушков, Е. Л. Ющенко. – Киев : Гостехиздат УССР, 1962. – 183 с. : ил.
2. Гнеденко Б. В. Элементы программирования / Б. В. Гнеденко, В. С. Королюк, Е. Л. Ющенко. – Москва : ГИФМЛ, 1961. – 348 с. : ил.
3. Дашевский Л. Н. Как это начиналось: Воспоминания о создании первой отечественной электронно-вычислительной машины МЭСМ [Электронный ресурс] / Л. Н. Дашевский, Е. А. Шкабара. – Москва : Новое в жизни, науке, технике. – Сер. Математика, кибернетика; № 1, 1981. – 64 с. – Режим доступа: kak_eto_nachinalos.pdf (computer-museum.ru) (19.05.2021).
4. Європейський віртуальний музей історії інформаційних технологій в Україні [Електронний ресурс]. – Режим доступу: http://www.icfcst.kiev.ua/MUSEUM/ (20.05.2021).
5. Іваненко Л. М. Основні питання побудови програмуючої програми для машини «Київ». Збірник праць ОЦ АН УРСР / Л. М. Іваненко, К. Л. Ющенко. – Т. II. – Київ, 1961.
6. Крещенко Т. О. Метод кластеризації з використанням багатовимірного адресного сортування / Т. Крещенко, Ю. Ющенко // Наукові записки НаУКМА. Комп'ютерні науки. – 2020. – Т.3.–С.83–87.https://doi.org/10.18523/2617-3808.2020.3.83-87
7. Ющенко Е. Л. Адресное программирование / Е. Л. Ющенко. – Киев : Гос. издательство технической литературы, УРСР, 1963. – 287 с. : ил.
8. Ющенко Ю. О. Багатовимірне впорядкування та його використання для вдосконалення інтерфейсу користувачів інформаційних систем / Ю. Ющенко // Наукові записки НаУКМА. Комп'ютерні науки. – 2018. – Т. 1. – С. 10–13. https://doi.org/10.18523/2617-3808.2018.10-13
9. 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 : іл.
10. Ющенко Ю. О. Окремі аспекти декларативності «мінус штрих-операції» / Ю. О. Ющенко // Наукові записки НаУКМА. Комп'ютерні науки. – 2020. – Т. 3. – С. 19–26. https://doi.org/10.18523/2617-3808.2020.3.17-26
11. Ющенко Ю. О. Катерина Логвинівна Ющенко – винахідниця Pointers та авторка однієї з перших в світі мов програмування високого рівня [Електронний ресурс] / Ю. О. Ющенко // Світ. – 2021. – № 5–6 (10.02). – С. 2–3. – Режим доступу: https://files.nas.gov.ua/PublicMessages/Documents/0/2021/02/210210172754893-2041.pdf (20.05.2021).
12. Videla Alvaro. Kateryna L. Yushchenko – Inventor of Pointers / Alvaro Videla // A Computer of One's Own – Pioneers of the Computing Age [Electronic resource]. – Mode of access: https://medium.com/a-computer-of-ones-own/kateryna-l-yushchenko-inventor-of-pointers-6f2796fa1798?fbclid=IwAR3fcqmC0COfy5EqyIHBrlQhCpno5MUFZjCUQ-SM-v-xhD0g3xbj_P2SRCM (20.05.2021).

References

Dashevskij, L. N., & Shkabara, E. A. (1981). *Kak jeto nachinalos': Vospominanija o sozdanii pervoj otechestvennoj jelektronno-vychislitel'noj mashiny MJeSM*. Moskva: Novoe v zhizni, nauke, tehnike [in Russian].

European Virtual Computer Museum. Development of Computer Science and Technologies in Ukraine. Retrieved from http://www.icfcst.kiev.ua/MUSEUM/.

Glushkov, V. M., & Jushhenko, E. L. (1962). *Vychislitel'naja mashina "Kiev". Matematicheskoe opisanie*. Kiev: Gostehizdat USSR [in Russian].

Gnedenko, B. V., Koroljuk, V. S., & Jushhenko, E. L. (1961). *Jelementy programmirovanija.* Moskva: HYFML [in Russian].

Ivanenko, L. M., & Yushchenko, K. L. (1961). *Osnovni pytannia pobudovy prohramuiuchoi prohramy dlia mashyny "Kyiv". Zbirnyk prats OTs AN URSR* (t. II). Kyiv [in Ukrainian].

Jushhenko, E. L. (1963). *Adresnoe programmirovanie*. Kyiv: Gos. izdatel'stvo tehnicheskoj literatury, URSR [in Russian].

Kreshchenko, T., & Yuschenko, Yu. (2020). Metod klasteryzatsii z vykorystanniam bahatovymirnoho adresnoho sortuvannia. *Naukovi zapysky NaUKMA. Kompiuterni nauky, 3,* 83–87 [in Ukrainian]. https://doi.org/10.18523/2617-3808.2020.3.83-87

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

Yuschenko, Yu. (2018). Bahatovymirne vporiadkuvannia ta yoho vykorystannia dlia vdoskonalennia interfeisu korystuvachiv informatsiinykh system. *Naukovi zapysky NaUKMA. Kompiuterni nauky, 1*, 10–13 [in Ukrainian]. https://doi.org/10.18523/2617-3808.2018.10-13

Yushchenko, Yu. O. (2020). Okremi aspekty deklaratyvnosti "minus shtrykh-operatsii". *Naukovi zapysky NaUKMA. Kompiuterni nauky, 3*, 19–26 [in Ukrainian]. https://doi.org/10.18523/2617-3808.2020.3.17-26

Yuschenko, 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].

Yuschenko, Yu. O. (2021). Kateryna Lohvynivna Yushchenko – vynakhidnytsia Pointers ta avtorka odniiei z pershykh v sviti mov prohramuvannia vysokoho rivnia. *Svit*, *5–6*, 2–3 [in Ukrainian]. Retrieved from https://files.nas.gov.ua/PublicMessages/Documents/0/2021/02/210210172754893-2041.pdf.

Yu. Yuschenko

TREE-SHAPED FORMATS OF ADDRESS PROGRAMMING LANGUAGE

In the Address Programming Language (1955), the concept of indirect addressing of higher ranks (Pointers) was introduced, which allows the arbitrary connection of the computer's RAM cells. This connection is based on standard sequences of the cell addresses in RAM and addressing sequences, which is determined by the programmer with indirect addressing. Two types of sequences allow programmers to determine an arbitrary connection of RAM cells with the arbitrary content: data, addresses, subroutines, program labels, etc. Therefore, the formed connections of cells can relate to each other. The result of connecting cells with the arbitrary content and any structure is called tree-shaped formats. Tree-shaped formats allow programmers to combine data into complex data structures that are like abstract data types. For tree-shaped formats, the concept of "review scheme" is defined, which is like the concept of "bypassing" trees. Programmers can define multiple overview diagrams for the one tree-shaped format. Programmers can create tree-shaped formats over the connected cells to define the desired overview schemes for these connected cells. The work gives a modern interpretation of the concept of tree-shaped formats in Address Programming. Tree-shaped formats are based on "stroke-operation" (pointer dereference), which was hardware implemented in the command system of computer "Kyiv". Group operations of modernization of computer "Kyiv" addresses accelerate the processing of tree-shaped formats and are designed as organized cycles, like those in high-level imperative programming languages. The commands of computer "Kyiv", due to operations with indirect addressing, have more capabilities than the first high-level programming language – Plankalkül. Machine commands of the computer "Kyiv" allow direct access to the i-th element of the "list" by its serial number in the same way as such access is obtained to the i-th element of the array by its index. Given examples of singly linked lists show the features of tree-shaped formats and their differences from abstract data types. The article opens a new branch of theoretical research, the purpose of which is to analyze the expediency of partial inclusion of Address Programming in modern programming languages.

Keywords: programming, pointers, dereference, dereference operator, indirection, multiply indirection, tree-shaped formats, Address Programming Language, Address Programming, IT history, programming history, computer architecture, abstract data types, lists, arrays, Ф-operation, computer "Kyiv".

Матеріал надійшов 22.05.2021