УДК 004.42
DOI https://doi.org/10.32782/2663-5941/2023.3.1/21

Дроменко В.Б.
Київський національний університет технологій та дизайну

Пилипенко Ю.М.
Київський національний університет технологій та дизайну

Кисельов В.Б.
Таврійський національний університет імені В.І. Вернадського

ЗАСТОСУВАННЯ РЕКУРСИВНИХ АЛГОРИТМІВ І ФУНКЦІЙ ПО ДЕКІЛЬКОХ ЗМІННИХ ПРИ ОРГАНІЗАЦІЇ ЦИКЛІЧНИХ ПРОЦЕСІВ

Метою роботи є пошук та реалізація засобів демонстрації особливостей рекурсивних обчислень по декількох змінних для організації складних циклічних процесів при розв'язувані задач.

Рекурсивний метод побудови (визначення) класу чи об'єкту заснований на визначені попередніх завдань одного чи декількох (зазвичай простих) його базових випадків чи методів та, в подальшому, заданням на їхній основі правила побудови класу, який визначається. Початкові значення задаються користувачем і вказуються правила побудови наступних образів через попередні. При дослідженнях використовуються теорія алгоритмів для структурованої побудови алгоритмів; поняття різних форм рекурсії та рекурсивних обчислень.

Продемонстровані нестандартні підходи застосування рекурсії по декількох змінних до розв'язку задач зі складними циклічними процесами, на відміну від традиційного застосування рекурсії по одній змінній. Реалізація цих підходів полягає у сформульованих математичних виразах рекурсивних співвідношень, в яких чітко визначені перелік параметрів, що передаються в рекурсивну функцію; формула наступного елементу, що використовуватиметься в рекурсивному процесі; умова припинення послідовності виконання рекурсивних викликів функції. Наведено приклади практичного застосування функції користувача (підпрограми) з рекурсією по двох та трьох змінних, які ґрунтуються на розроблених рекурсивних співвідношеннях.

Запропоновані підходи демонстрації особливостей рекурсивних обчислень для організації складних циклічних процесів ґрунтуються на розроблених рекурсивних співвідношеннях і реалізовано у вигляді функції користувача (підпрограми) з рекурсією по двох та трьох змінних.

Наведені описи алгоритмів на основі рекурсивних співвідношень по декількох змінних дають можливість наочної уяви про реалізацію складних циклічних процесів і не потребують традиційного зображення алгоритмів у вигляді блок-схем. При цьому, запропоновані алгоритми можуть бути реалізовані у функціональному програмуванні багатьма сучасними мовами програмування, оскільки вони дають змогу компонувати та комбінувати функції.

Ключові слова: *рекурсія, рекурсивні співвідношення, функція користувача (підпрограма) з рекурсією по декількох змінних, програмна реалізація рекурсії по декількох змінних.*

Постановка проблеми. Застосування рекурсії є потужним методом при розв'язувані цілої низки задач, серед яких реалізація алгоритмів обходу списків, дерев, чисел Фібоначчі, аналізаторів виразів, комбінаторних задач, тощо [1–2].

У більшості розробників виникають певні труднощі, коли вони вперше стикаються з такою алгоритмічною структурою, як рекурсивні обчислення. Під рекурсивними функціями розуміють функції, які при своєму визначені звертаються самі до себе. Відомо, що рекурсія може бути: прямою – якщо функція в тілі містить тільки виклик самої себе; та непрямою – якщо функція викликає іншу функцію, а та, у свою чергу, викликає першу [3–4].

Аналіз останніх досліджень і публікацій. При кожному рекурсивному виклику всі попередні значення внутрішніх локальних змінних та переданих у функцію користувача параметрів зберігаються в пам'яті. Обов'язковою умовою виконання наступного кроку рекурсивного виклику є необхідність змінення значення хоча б одного з параметрів функції. Процес рекурсивних викликів функції припиняється в разі досягнення змінюваного параметру певного кінцевого значення, наприклад, оброблено останній елемент в масиві [5–7].

Рекурсивне звернення до функції може бути здійснене тільки в разі, якщо алгоритм визначений (побудований) рекурсивно.

Рекурсію достатньо часто порівнюють з ітерацією. Будь-яку рекурсивну функцію можна визначити і нерекурсивно, тобто ітеративно через цикли, що є доведеним фактом. Також цілком доведеним є і зворотне твердження: будь-яку нерекурсивну функцію можна визначити рекурсивно. Це означає, що рекурсивність не є властивістю самої функції, а є лише властивістю її опису [8–9].

Виникає логічне питання: а який опис – рекурсивний чи ітеративний – краще застосовувати у програмуванні? Однозначної відповіді немає. Але, в загальному випадку рекурсивний опис функції, як правило, більш компактніший, а ітеративний – менш компактніший. При виборі способу опису функції потрібно враховувати, що обчислення рекурсивних функцій, зазвичай, відбувається довше і використовує більше пам'яті, ніж ітеративних (втрати відбуваються за рахунок повторних викликів, введення нових локальних змінних і т. і.) [2, 4, 7].

Проте, організація циклічного процесу за допомогою рекурсії має певні переваги, серед яких можна виділити наступні:

1) можливість представлення у природний спосіб складних, на перший погляд, алгоритмів;

2) рекурсивні алгоритми є більш наочними у порівнянні з ітераційними;

3) рекурсію простіше реалізувати ніж ітерацію для багатьох поширених задач.

Однак, в більшості відомих прикладів широко застосовується рекурсія по одній змінній [1, 3, 10]. В той же час, рекурсія може бути значно корисніщою, якщо її застосувати по двох чи більше аргументах одночасно. Тому, пошук та реалізація засобів демонстрації особливостей рекурсивних обчислень по декількох змінних для організації складних циклічних процесів при розв'язувані задач є актуальними.

Постановка завдання. Спробуємо продемонструвати особливості рекурсивних обчислень по декількох змінних для організації складних циклічних процесів, і рекурсія стане зрозумілим і корисним інструментом в арсеналі програмування.

Щоб типовий циклічний процес перетворити на рекурсивний, необхідно правильно і чітко виділити (визначити) такі важливі моменти:

1) умову припинення послідовності виконання рекурсивних викликів функції;

2) формулу наступного елементу, що використовуватиметься в рекурсивному процесі;

3) перелік параметрів, що передаються в рекурсивну функцію. Один з параметрів обов'язково є лічильником, який змінює своє значення. Інші параметри є додатковими, наприклад, посилання на масив, обробка якого здійснюється.

Враховуючи вищезазначене, покажемо приклад застосування алгоритму і функції користувача (підпрограми) з рекурсією по декількох змінних.

Виклад основного матеріалу дослідження. Розглянемо на прикладах, використовуючи принцип «від простого до складного», відомі типові задачі, в яких застосовуються рекурсивні функції при побудові різних алгоритмів. Всі приклади ми будемо ілюструвати фрагментами програмного коду мови програмування С++ в середовищі Microsoft Visual Studio 2019.

При невірному застосуванні рекурсія є недоцільною, адже обчислення проводяться частіше, ніж при ітерації. Щоб уникнути подібних помилок з рекурсивними функціями, почнемо з найпростіших задач.

Типовий приклад 1.

Знайти суму $S = \sum_{i=1}^{n} \frac{i+1}{i^2+1}$, де значення n наперед відоме.

Наведений на рис. 1 фрагмент програмного коду демонструє ітеративний підрахунок суми S. Вважатимемо, що значення параметру n є визначеним у попередніх рядках програмного коду

```
S=0;
for (i=1; i<=n; i++)
      S = S + (i+1)/(pow(i,2)+1);
```

Рис. 1. Фрагмент програмного коду підрахунку суми

Типовий приклад 2.

З іншого боку, дії в задачі типового прикладу 1 повторюються, змінюються лише параметри, тому можна її розв'язати і наступним чином, застосовуючи рекурсивну функцію S_n, яка може бути визначена двома наступними математичними виразами рекурсивних співвідношень:

$$S_n = S_{n-1} + \frac{n+1}{n^2+1}; \quad (1)$$

$$S_0 = 0. \quad (2)$$

Фрагмент програмного коду по створенню функції S_n наведений на рис. 2.

```
double Function(double n)
{
     if (!n)
          return 0;
     else
          return Function (n-1) + (n+1)/(pow(n,2)+1);
}
```

Рис. 2. Фрагмент програмного коду підрахунку суми із застосуванням рекурсивної функції

Результат розрахунку значення суми S за рекурсивними співвідношеннями (1) або (2), в залежності від переданого у функцію користувача значення n, буде повернуто в основну програму.

Типовий приклад 3.

Необхідно знайти значення наступного виразу:

$$F(n,a) = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \ldots \cfrac{1}{1 + \cfrac{1}{a}}}}}$$

де a – задане додатне число, а дріб містить n «поверхів».

Таку задачу достатньо просто можна розв'язати використовуючи рекурсію, беручи до уваги наступні два математичні вирази рекурсивних співвідношень, які дають уяву про алгоритм розв'язку задачі, а саме:

$$F(1,a) = 1 + \frac{1}{a}, \qquad (3)$$

$$F(n,a) = 1 + \frac{1}{F(n-1,a)}. \qquad (4)$$

При цьому, немає потреби традиційного зображення алгоритмів у вигляді блок-схем, оскільки у наведених рекурсивних співвідношеннях чітко визначені перелік параметрів, що передаються в рекурсивну функцію; формула наступного елементу, що використовуватиметься в рекурсивному процесі; умова припинення послідовності виконання рекурсивних викликів функції.

Фрагмент програмного коду по створенню функції користувача, яка реалізує поставлену задачу наведено на рис. 3.

```
double Function(int n, double a)
{
if (n == 1)
      return 1 + 1 / a;
else
      return 1 + 1 / Function(n - 1,a);
}
```

Рис. 3. Фрагмент програмного коду із застосуванням рекурсивної функції по одній змінній

Результат розрахунку значення суми F за рекурсивними співвідношеннями (3) або (4), в залежності від переданого у функцію користувача значення n, буде повернуто в основну програму.

Типові приклади 2 та 3 демонструють застосування рекурсії по одній змінній, яка набула значного поширення при розв'язку різноманітних обчислювальних задач. Вже на цих простих прикладах застосування рекурсивної функції по одній змінній можна спостерігати характерну особливість рекурсивної функції. При визначенні такої функції спочатку поступово спрощується її аргумент, доки не дійдемо до найпростішого аргумента, при якому функція видає відповідь у явному вигляді без повторного звернення до самої себе. В подальшому починається рух у зворотному боці, обчислюючи значення функції для дедалі складніших аргументів.

Покажемо на наступному прикладі, як для вирішення більш складних задач нами застосований алгоритм з рекурсією по двох змінних.

Приклад 4.

Знайти значення суми

$$S = \sum_{i=1}^{n} 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \ldots \cfrac{1}{1 + \cfrac{1}{a}}}}}$$

де *i*-ий доданок містить *i* «поверхів»; параметр a – довільне додатне число.

Пояснимо роботу алгоритму функції користувача (підпрограми) recursion(k, n, a), в якій використовуватимемо рекурсію по двох перших параметрах (k та n). Третій параметр a – довільне додатне число.

Вважатимемо, що основна комп'ютерна програма виконує стандартне введення всіх потрібних початкових даних для розрахунку, звернення до функції користувача (підпрограми) з передачею значень параметрів k, n, a в функцію recursion з основного алгоритму та виведення результатів розрахунку.

При розв'язку задачі алгоритм побудовано таким чином, що параметр k прийматиме лише два значення – «0» або «1». Значення «0» застосовується для підрахунку чергового (одного) значення доданку суми S, значення «1» – для формування накопичення суми.

В разі, коли значення n кількості «поверхів» дорівнює 1 (n = 1), виникає окремий випадок, при якому значення суми S містить лише один доданок і визначатиметься наступним рекурсивним співвідношенням

$$\text{recursion}(0,1,a) = 1 + \frac{1}{a}. \qquad (5)$$

В разі, коли значення n кількості «поверхів» відмінно від 1, підрахунок одного значення доданку суми S з n «поверхів» визначатиметься рекурсивним співвідношенням

$$\text{recursion}(0,n,a) = 1 + \frac{1}{\text{recursion}(0,n-1,a)} \qquad (6)$$

з викликом функції recursion( ) для виразу з (n – 1) «поверхів».

В разі, коли параметр k набуває значення «1», відбувається формування накопичення суми.

При кількості «поверхів», яке дорівнюватиме 0 (n = 0) виникає окремий випадок, в результаті чого значення суми S визначатиметься рекурсивним співвідношенням

$$\text{recursion}(1,0,a) = 0\,. \qquad (7)$$

В разі, коли значення n кількості «поверхів» відмінно від 0, формування накопичення суми S при n «поверхах» визначатиметься рекурсивним співвідношенням

$$\text{recursion}(1,n,a) = \text{recursion}(1,n-1,a) + \text{recursion}(0,n,a) \qquad (8)$$

з виконанням рекурсивного виклику функції recursion( ).

Наведемо приклад програмної реалізації цього алгоритму. Фрагмент комп'ютерної програми з реалізацією рекурсивної функції по двох змінних представлено на рис. 4 розробленою авторами функцією користувача (підпрограмою).

Результат розрахунку значення суми S за рекурсивними співвідношеннями (5) – (8), в залежності від переданих у функцію користувача значень k та n, які можуть спричинити вищезазначені окремі випадки, буде повернуто в основну програму.

```
double recursion(bool k, int n, double a)
{
	if (k == false)
	{
		if (n == 1) return  1 + 1 / a;
		else return 1 + 1 / recursion(0, n - 1, a);
	}
	else
	{
		if (!n) return 0;
		else return recursion(1, n - 1, a) + recursion(0, n, a);
	}
}
```

Рис. 4. Функція користувача (підпрограма) з реалізацією рекурсії по двох змінних

Приклад 5.

Знайти значення виразу $S = \sum_{i=1}^{m} \prod_{j=1}^{n} a_{ij}$, де числа a_{ij} ($i = 1, 2,\ldots, m$, $j = 1, 2,\ldots, n$) задані.

Пояснимо роботу алгоритму функції користувача (підпрограми) recursion(k, m, n), в якій використовуватимемо рекурсію по трьох параметрах.

При розв'язку задачі алгоритм побудовано так само, як в прикладі 4, де параметр k прийматиме лише два значення – «0» або «1». Значення «0» застосовується для підрахунку чергового (одного) значення доданку суми S, яке, в свою чергу, є добутком чисел a_{ij}; значення «1» – для формування накопичення суми.

Початкове значення для накопичення добутку (n = 0) визначатиметься рекурсивним співвідношенням

$$\text{recursion}(0,m,0) = 1\,. \qquad (9)$$

Початкове значення для накопичення суми (m = 0) визначатиметься рекурсивним співвідношенням

$$\text{recursion}(1,0,n) = 0\,. \qquad (10)$$

Для знаходження *m*-го доданку суми, яке по суті є накопиченим добутком множників заданих чисел a_{ij}, застосуємо рекурсивне співвідношення

$$\text{recursion}(0,m,n) = \text{recursion}(0,m,n-1) \cdot a_{mn} \qquad (11)$$

з викликом функції recursion( ) для попереднього значення накопиченого добутку.

Формування накопичення суми визначатиметься наступним рекурсивним співвідношенням з викликом функції recursion( )

$$\text{recursion}(1,m,n) = \text{recursion}(1,m-1,n) + \text{recursion}(0,m,n)\,. \qquad (12)$$

Фрагмент комп'ютерної програми з реалізацією рекурсивної функції, в даному разі по трьох змінних, для знаходження значення вищенаведеного прикладу представлено на рис. 5 розробленою авторами функцією користувача (підпрограмою).

```
double recursion(bool k, int m, int n)
{
	if (k == false)
	{
		if (!n) return 1;
		else return recursion(0, m, n - 1) * a[m - 1][n - 1];
	}
	else
	{
		if (!m) return 0;
		else return recursion(1, m - 1, n) + recursion(0, m, n);
	}
}
```

Рис. 5. Функція користувача (підпрограма) з реалізацією рекурсії по трьох змінних

Результат розрахунку значення суми S за рекурсивними співвідношеннями (9)–(12), в залежності від переданих у функцію користувача значень k, m та n, буде повернуто в основну програму.

На прикладах 4 та 5, які реалізують рекурсію по декількох змінних, продемонстровано, що опис рекурсивної функції містить дві складові:

1) нерекурсивну гілку, де передбачені найпростіші випадки, при яких функція видає відповідь у явному вигляді. Зазначимо, що наявність хоча б однієї нерекурсивної гілки є обов'язковою. Інакше при обчисленні функції ланцюжок рекурсивних викликів виявиться формально нескінченно довгим, а простіше кажучи – не відбудеться вихід з рекурсії;

2) рекурсивну гілку, де передбачено загальний випадок розв'язку задачі. Загальний випадок зводиться до простіших аналогічних випадків, для вирішення яких рекурсивно застосовується та сама функція, а потім з отриманих відповідей утворюється остаточна відповідь.

Зазначимо, що наведені приклади з запропонованими рекурсивними співвідношення не при-

мушують розробників програм застосовувати рекурсивний алгоритм в якості «обов'язкового» при розв'язувані задач зі складними циклічними процесами. Автори показали один з підходів для розв'язку задач такого типу, а як розв'язувати ту чи іншу конкретну задачу – це прерогатива розробника. Однак, на наш погляд, наведена інформація буде корисною як починаючим розробникам, так і науково-педагогічним працівникам.

Висновки. Таким чином, продемонстровано особливості застосування рекурсивних функцій по декількох змінних, які можна використовувати при організації достатньо складних циклічних процесів. Показано, що коли поставлену задачу можна розбити на більш дрібні підзадачі, які можна розв'язати одним і тим самим методом, рекурсивні алгоритми стануть гарним вибором. Крім того, рекурсивні співвідношення дають можливість наочної уяви про алгоритм реалізації складних циклічних процесів на основі рекурсивних обчислень по декількох змінних і не потребують традиційного зображення алгоритмів у вигляді блок-схем. Наведені фрагменти програм з побудови відповідних рекурсивних функцій демонструють переходи за запропонованими рекурсивними співвідношенням по різних аргументах та поєднують в єдине ціле функцію для знаходження кінцевого результату. Запропоновані алгоритми рекурсивних функцій по декількох змінних можуть бути реалізовані у функціональному програмуванні багатьма сучасними мовами програмування, оскільки вони дають змогу компонувати та комбінувати функції.

Список літератури:

1. Трофименко О. Г. С++. Основи програмування. Теорія та практика :підручник / [О. Г. Трофименко, Ю. В. Прокоп, І. Г. Швайко, Л. М. Буката та ін.] ; за ред. О. Г. Трофименко. – Одеса: Фенікс, 2010. – 544 с.
2. Stroustrup, Bjarne (2008). The C++ Programming Language (Fourth ed.). *Addison-Wesley.*
3. Васильєв О. М. Програмування на С++ в прикладах і задачах / О. М. Васильєв – Київ : Ліра-К, 2017. – 382 с.
4. Lippman, Stanley B.; Lajoie, Josée; Moo, Barbara E. (2013). C++ Primer (Fifth ed.). *Addison-Wesley.*
5. Stroustrup, Bjarne (2014). Programming: Principles and Practice Using C++ (Second ed.). *Addison-Wesley.*
6. Josuttis, Nicolai M. (2012). The C++ Standard Library, A Tutorial and Reference (Second ed.). *Addison-Wesley.*
7. Steven Prata (2012). C++ Primer Plus. *(*6th *ed.). Addison-Wesley.*
8. Грицюк Ю. І. Програмування мовою С++ : навчальний посібник / Ю. І. Грицюк, Т. Є. Рак – Львів : Вид-во Львівського ДУ БЖД, 2011. – 292 с.
9. Scott Meyers (2008). Effective C++, Third Edition: 55 Specific Ways to Improve Your Programs and Designs, (3rd Edition).
10. Васильєв О. М. Алгоритми / О. М. Васильєв – Київ : Ліра-К, 2022. – 424 с.

Dromenko V.B., Pylypenko Yu.M., Kyselov V.B. APPLICATION OF RECURSIVE ALGORITHMS AND FUNCTIONS ACCORDING TO SEVERAL VARIABLES IN THE ORGANIZATION OF CYCLIC PROCESSES

The purpose of the work is to find and implement means of demonstrating the features of recursive calculations on several variables for the organization of complex cyclical processes when solving problems.

The recursive method of building (defining) a class or object is based on the defined preliminary tasks of one or more (usually simple) of its basic cases or methods and, subsequently, the task of building rules of the defined class based on them. The initial values are set by the user and the rules for building subsequent images through the previous ones are specified. The research uses the theory of algorithms for the structured construction of algorithms; concepts of various forms of recursion and recursive calculations.

Non-standard approaches of applying recursion on several variables to the solution of problems with complex cyclic processes, in contrast to the traditional application of recursion on one variable, are demonstrated. The implementation of these approaches consists in formulated mathematical expressions of recursive relations, in which the list of parameters transferred to the recursive function is clearly defined; the formula of the next element to be used in the recursive process; the condition for terminating the sequence of execution of recursive function calls. Examples of the practical application of the user function (subroutine) with recursion in two and three variables are given, which are based on the developed recursive relations.

The proposed approaches for demonstrating the features of recursive calculations for the organization of complex cyclic processes are based on the developed recursive relations and are implemented in the form of a user function (subroutine) with recursion in two and three variables.

The given descriptions of algorithms based on recursive relations for several variables provide a visual representation of the implementation of complex cyclic processes and do not require the traditional representation of algorithms in the form of block diagrams. At the same time, the proposed algorithms can be implemented in functional programming in many modern programming languages, as they allow to compose and combine functions.

Key words: *recursion, recursive relations, user function (subroutine) with recursion on several variables, software implementation of recursion on several variables.*