УДК 519.683.2
DOI https://doi.org/10.15589/znp2023.1(490).23

LEARNING PROGRAMMING USING STRUCTURED BLOCK DIAGRAMS AND DESIGN TEMPLATES

НАВЧАННЯ ПРОГРАМУВАННЮ З ЗАСТОСУВАННЯМ СТРУКТУРОВАНИХ БЛОК–СХЕМ ТА ШАБЛОНІВ ПРОЕКТУВАННЯ

Vasyl S. Kostyrko
vkostyrko@lute.lviv.ua
ORCID: 0000-0002-6366-8695

Anatolij V. Kostenko
avak54@lute.lviv.ua
ORCID: 0000-0002-2162-7852

Mykhaylo I. Plesha
milan@lute.lviv.ua
ORCID: 0009-0002-6496-1102

В. С. Костирко,
канд. ф.-м. наук, доцент

А. В. Костенко,
канд. ф.-м. наук, доцент

М. І. Плеша,
канд. ф.-м. наук, доцент

Lviv University of Trade and Economics, Lviv
Львівський торговельно–економічний університет, м. Львів

Abstract. Despite the general belief in the advantages of structural programming and top-down programming technologies, traditional block diagrams continue to be used to describe algorithms [1, 2]. **The purpose** of the article is to overcome contradictions between the algorithmic and programmatic levels of process representation, which is especially noticeable in programming teaching. **Scientific novelty**. Modern programming languages are based on structured statements of conditionals, loops, switches, etc., replacing direct transitions. However, the outdated formalism of flowcharts continues to be used to describe algorithms, which led to dangerous direct transitions (related to goto statements) when moving to programs and prevented the use of such advanced technologies as top-down programming and program verification. Perhaps this has its explanation in the obviousness of the construction of conventional flowcharts, teaching traditions, etc. However, such methods of describing algorithms as UML-diagrams were not accepted as a formalism for writing algorithms and teaching programming. In this work, several templates are proposed that allow expanding the construction of block diagrams and making them suitable for adequate description of structured programs. Among them, compositions of complete and incomplete branching, loops with precondition and postcondition, loops on a sequence, as well as a switchs are considered. Several technologies for structuring programs (transition from conventional to structured flowcharts) are described. One of the technologies named introduction of additional arithmetic variables can be applied automatically. The computational equivalence of block diagrams structured in this way to the initial ones is proven, in the sense that they generate the same computational process. The proof is based on the concept of an illuminated flowchart. **Conclusion:** the use of structured flowcharts in teaching programming demonstrated the perspective of this approach. This actualizes the task of developing appropriate graphical tools, as well as APIs for generating programs in popular programming languages.
Key words: program; flowchart; design template; structural operator; teaching programming; structural programming.

Анотація. Незважаючи на загальне визнання переваг технологій структурного програмування та програмування методом зверху вниз, для опису алгоритмів продовжують застосовувати традиційні блок–схеми [1, 2]. **Метою статті** є подолання суперечностей між алгоритмічним та програмним рівнями представлення процесів, яке особливо відчутне при навчанні програмуванню. **Наукова новизна**. Сучасні мови програмування базуються на структурованих операторах умовних, циклів, перемикачів тощо, заміщаючи ними прямі переходи. Однак, для опису алгоритмів продовжує застосовуватися застарілий формалізм блок–схем, який при переході до програм приводив до небезпечних прямих переходів (пов'язаних з операторами goto) і перешкоджав застосуванню таких прогресивних технологій, як програмування зверху вниз та верифікація програм. Можливо, це має своє пояснення в очевидності конструкції звичайних блок–схем, традиціях навчання тощо. Разом з тим, такі способи опису алгоритмів як UML–діаграми, не були сприйняті як формалізм для запису алгоритмів та навчання програмуванню. В даній роботі запропоновано декілька шаблонів, які дозволяють розширити конструкцію

блок–схем і зробити їх придатними для адекватного опису структурованих програм. Серед них розглянуто композиції повного та неповного розгалуження, циклу з передумовою та післяумовою, циклу по послідовності, а також перемикача. Описано декілька технологій структурування програм (переходу від звичайних блок–схем до структурованих). Показано, що одна з технологій – введення додаткових арифметичних змінних – може застосовуватися автоматично. Доведена обчислювальна еквівалентність структурованих таким чином блок–схем початковим, в тому смислі, що вони породжують той же обчилювальний процес. Доведення базується на понятті просвітленої блок–схеми. **Висновок:** застосування структурованих блок–схем в навчанні програмуванню продемонструвало перспективність цього підходу. Це актуалізує задачу розробки відповідних графічних інструментів, а також API для генерації програм у популярних мовах програмування.
Ключові слова: програма; блок–схема; шаблон проектування; структурний оператор; навчання програмуванню; структурне програмування.

АКТУАЛЬНІСТЬ ПРОБЛЕМИ

Блок–схемою назвемо орієнтований граф, вузли якого відповідають діям алгоритму, а дуги – можливим переходам управління від однієї дії до іншої. У вузлах блок–схеми описуються дії, а дуги позначаються умовами переходу між блоками. Вузли блок–схеми позначаються фігурами, форма яких визначається типом представлених ними дій, а дуги – стрілками або лініями. Так, наприклад, вузол з присвоєнням значень змінним позначається прямокутником, вузол з умовою переходу – ромбом, а вузол для виводу значень виразів – паралелограмом.

Зображені таким чином вузли називаються блоками. На переходи між блоками не накладається жодних обмежень, що призводить до заплутаних та незрозумілих структур. В результаті побудовані за ними програми мають настільки ж заплутану та незрозумілу структуру, що ускладнює їх налагодження та супровід.

Оператори циклу в блок–схемах не мають спеціальних блоків і реалізуються блоками присвоювання та умовними, а також належними переходами. Блок–схеми не мають ієрархій, тому не мають і позначення для тіла циклу. Кожен блок має один вхід і один або декілька виходів. Блок–схеми принципово неструктуровані, оскільки стрілку від будь–якого виходу блока можна протягнути до входу будь–якого блока.

Таким чином, виникає суперечність між неструктурованою формою блок–схеми алгоритму та ніби відповідною йому структурованою програмою. Ця суперечність приводить до концептуального розриву між алгоритмом та відповідною програмою, породжуючи помилки в програмах, ускладнюючи їх розробку, тестування, верифікацію та подальшу підтримку.

АНАЛІЗ ОСТАННІХ ДОСЛІДЖЕНЬ І ПУБЛІКАЦІЙ

Ця суперечність досліджувалась у роботах [3, 4] та інших. Для порівняння неструктурованих блок–схем та структурованих програм їх потрібно було б привести до “спільного знаменника”: додати в мову блок–схем структуровані блоки (і порівнювати їх на рівні блок–схем) або зберегти в мовах програмування оператор переходу (і порівнювати їх на рівні мов програмування). Другий підхід самим розвитком технології програмування був відкинутий як безперспективний. Тому зупинимося на першому.

Квінтесенція досліджень [3, 4] така: неструктуровану блок–схему (програму) неможливо перетворити в структуровану, якщо базуватися на тому ж алгоритмі, на тих же блоках (операторах) та не вводити додаткових змінних. Перетворення неструктурованої блок–схеми в структуровану назвемо структуруванням.

Метою статті є подолання суперечності між представленням алгоритмів блок–схемами та їх програмною реалізацією сучасними мовами програмування. Ця суперечність особливо відчутна при навчанні програмуванню, оскільки для наочності алгоритми прийнято представляти блок–схемами.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

В [4] та [5] розглянуто декілька методів структурування блок–схем. Ці методи зводяться до наступного:

– Дублювання тих фрагментів блок–схем, які перешкоджають структурованості;

– Введення додаткових змінних, блоків присвоювання та перевірки значень цих змінних.

Якщо з методом дублювання все зрозуміло, то метод введення додаткових змінних потребує коментарів. Тут застосовуються додаткові змінні двох типів:

– Арифметичні змінні стану, які визначають номер наступного оператора у виконанні блок–схеми;

– Логічні змінні, які визначають напрямок руху виконання.

Перший метод достатньо універсальний і може застосовуватися автоматично, в той час як другий вимагає творчого підходу. Однак, програма, побудована в результаті застосування змінних стану, повністю втрачає свою первісну структуру і тому цей метод застосовується переважно в теоретичних дослідженнях. Метод логічних змінних дозволяє частково зберегти структуру первісної програми, тому може застосовуватися програмістами і в практичній діяльності. Цей метод дозволяє програмісту зрозуміти, чого не вистачає програмі для того, щоб бути структурова-

ною, і таким чином вчить створювати структуровану програму з самого початку її проектування.

При встановленні обмежень на можливі переходи між блоками блок–схем потрібно переконатися в тому, що ці обмеження не завадять реалізувати будь–який алгоритм. Таке твердження можна формалізувати і довести математично. Воно відоме як теорема про представлення [1, 5].

Розглянемо базові конструкції запропонованих нами структурованих блок–схем. Від традиційних блок–схем до структурованих перенесемо елементарні блоки (процесу, вводу–виводу, підалгоритму та пуску–зупинки), а також лінії потоку (табл. 1).

Зауважимо, що при переході до структурованих блок–схем з нотації блок–схем усунуто позначений ромбом блок рішення. Зате сюди введено ієрархію, яка реалізується операціями композиції. Лінії потоку тут не грають самостійної ролі і застосовуються лише в рамках композицій.

Згідно з [2] назвемо блок–схему простою, якщо вона має один вхід та один вихід. Відмітимо, що базові блоки мови блок–схем (процесу, вводу/виводу та під алгоритму) можна розглядати як елементарні прості блок–схеми. Надалі там, де не виникає неоднозначності, базові блоки та прості блок–схеми будемо зображати прямокутниками, абстрагуючись від їх рівня та призначення. Також не будемо зображати блоків початку та кінця процесу.

Ієрархію в мову структурованих блок–схем привносять операції композиції таких трьох видів:

– лінійна;
– розгалуження;
– повторення.

Аргументами операцій композиції служать структуровані блок–схеми та умови. Блоки процесу, вводу/виводу та підалгоритму є елементарними структурованими блок–схемами.

Неважко переконатися в тому, що всі операції композиції утворюють прості блок–схеми. Тому результат застосування кожної операції композиції можна розглядати як блок наступного рівня ієрархії.

На рис. 1 наведено результат застосування композиції послідовності до структурованих блок–схем S_1, S_2, …, S_n.

Лінії потоку показують послідовність виконання блоків (блок–схем), а пунктирна лінія обмежує композицію. На рис. 2 наведено композицію розгалуження, а на рис. 3 – композицію повторення.

Аргументами композиції розгалуження служать структуровані блок–схеми *A*, *B* та умова *L*. Причому, ромб не є блоком, а лише елементом оформлення композиції. Аргументами повторення служать структурована блок–схема *T* (тіло циклу) та умова *L* повто-

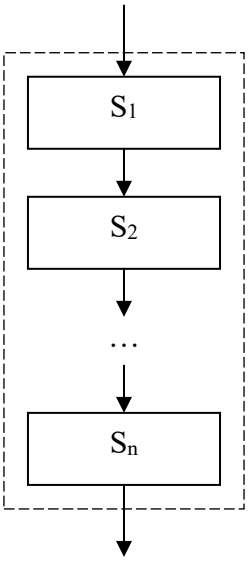
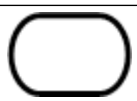
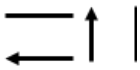


Рис. 1. Лінійна композиція

Таблиця 1. Базові конструкції блок–схем

Назва	Зображення	Призначення
Процес		Виконання операції або групи операцій, завдяки яким змінюються значення, форми представлення або розташування даних
Ввід/вивід		Перетворення даних у форму, придатну для обробки (ввід) або відображення результатів (вивід)
Підалгоритм		Виконання іншого алгоритму як базової операції
Пуск/зупинка		Початок, кінець, переривання процесу виконання
Лінія потоку		Визначення можливої послідовності блоків під час виконання

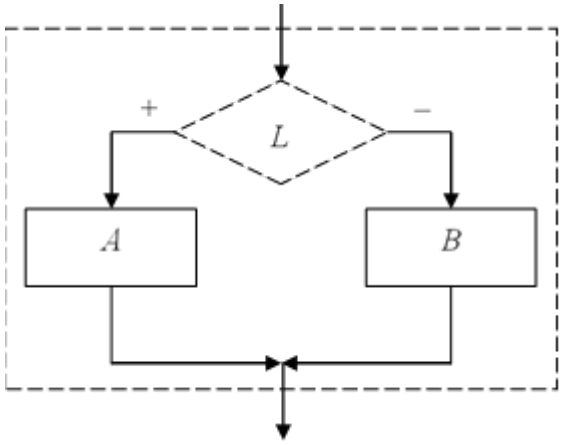


Рис. 2. Композиція розгалуження

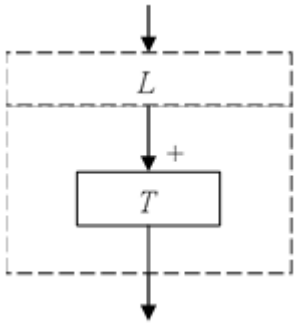


Рис. 3. Композиція повторення

рення тіла циклу. Пунктирні лінія на рисунках визначають границі композицій.

Всі три види композицій утворюють прості блок–схеми, які можуть служити аргументи композицій наступного рівня ієрархії.

Виконанням блок–схеми назвемо послідовність блоків та відмічених умовами переходів стрілок. Довжиною виконання блок–схеми назвемо кількість блоків у ньому. Початком виконання блок–схеми є блок пуску, а кінцем – блок зупинки (тоді виконання буде завершеним) або стрілка виходу з блоку (незавершене виконання).

Дерево виконання блок–схеми відображає всі можливі її виконання, одержані реалізацією умовних розгалужень. Висотою дерева виконання назвемо максимальну довжину його виконань. Наприклад, на рис. 4 зображено дерево виконання блок–схеми, наведеної на рис. 3, висотою 3.

Для однозначності представлення будемо вважати, що на дереві виконання ліва стрілка позначається істинною умовою переходу, а права – фальшивою. Блок–схеми назвемо структурно еквівалентними, якщо вони породжують одне дерево виконання.

Постає запитання, чи будь–який алгоритм можна реалізувати в описаній мові структурованих блок–схем? Стверджується, що так, якщо його можна представити простою блок–схемою. Покажемо, що кожну просту блок–схему можна формально перетворити на еквівалентну структуровану. Для доведення цього факту скористаємося методом введення змінних стану [5, 2] та методом математичної індукції.

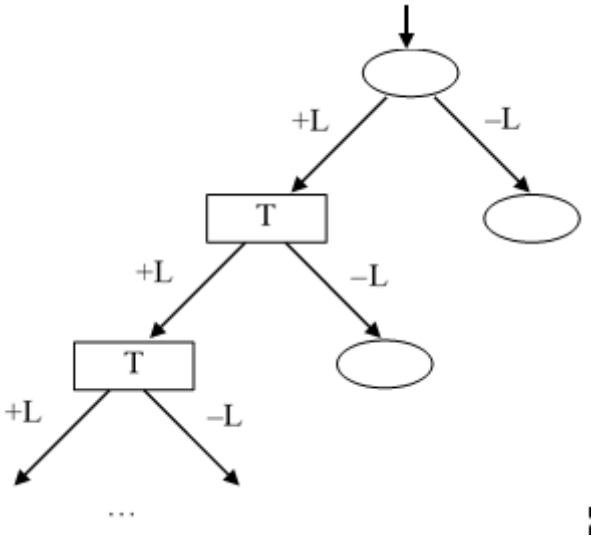


Рис. 4. Дерево виконання блок–схеми

Розглянемо довільну просту блок–схему S і наступним чином побудуємо структуровану блок–схему $C(S)$. Для цього пронумеруємо всі блоки схеми S (окрім блоку зупинки). Кількість блоків позначимо через n, а блок з номером i – через S_i. Без обмеження загальності можемо вважати, що вхідна стрілка блок–схеми веде в блок з номером 1. Будемо вважати, що стрілки, які виходять з блоків рішення, позначені символами “+” (умова переходу виконується) та “–” (умова переходу не виконується).

Введемо додаткову цілочисельну змінну V, яка прийматиме значення з інтервалу $[1, n+1]$. За допомогою композицій послідовності та повторення побудуємо таку структуровану блок–схему (рис. 5). Під час виконання змінна V буде зберігати інформацію про блок, який повинен виконуватися наступним. Блок H буде аналізувати значення змінної стану і залежно від її значення виконувати один з блоків S_1, S_2,… S_n. Замість того, щоб перейти до виконання іншого блоку, змінній стану присвоюється його номер – і цикл повторюється.

Якщо S_i є простим блоком, то з нього виходить лише одна стрілка; через c_i позначимо номер блоку, до якого вона веде; рис. 7а демонструє вигляд відповідного блоку U_i.

Якщо S_i є блоком рішення, то з нього виходять дві стрілки; через d_i позначимо номер блоку, до якого веде стрілка із значком “+”, а через e_i – номер блоку, до яких веде стрілка із значком “–”; рис. 7б демонструє вигляд відповідного блоку U_i.

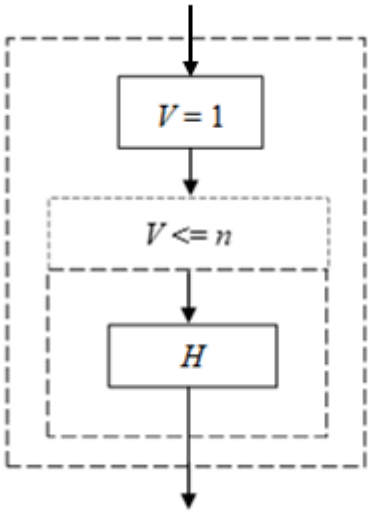


Рис. 5. Структурований еквівалент простої блок–схеми

Якщо стрілка веде до виходу з блок–схеми, то відповідна їй величина c_i, d_i чи e_i буде рівна $n + 1$.

В дереві виконання структурованої блок–схеми додатковими елементами назвемо блоки присвоювання значень додатковій змінній та розгалуження, породжені композиціями (з умовами переходу, які залежать від додаткової змінної). При видаленні з нього додаткових елементів утворюється дерево, яке назвемо освітленим.

Теорему представлення сформулюємо у вигляді індуктивного твердження *T(m)*: будь–яке дерево виконання довільної простої блок–схеми *S,* висота якого не перевищує *m,* співпадає з освітленим деревом виконання структурованої описаним способом блок–схеми *C(S)* такої ж висоти.

Твердження *T(1)* очевидне, оскільки обидва дерева складаються лише з одного блоку – пуску. Припустимо, що для довільного цілого додатного числа *m*

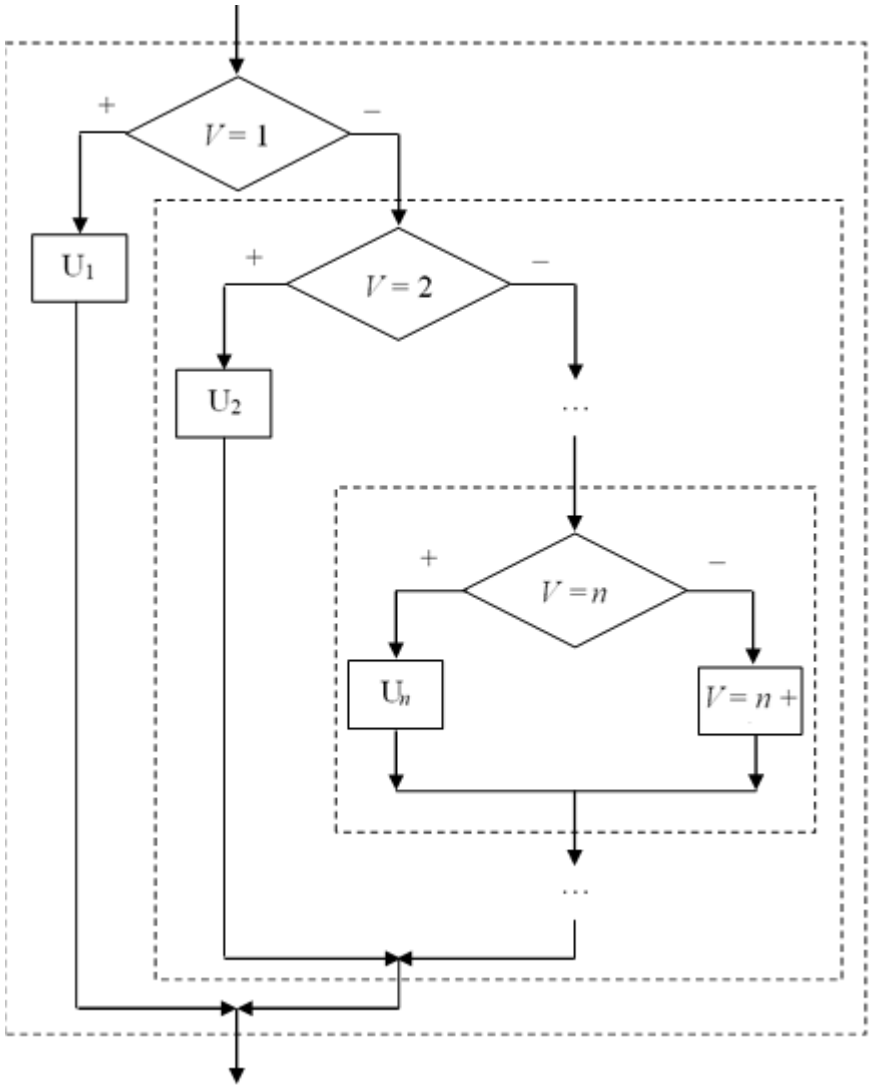


Рис. 6. Тіло структурованого еквіваленту блок–схеми

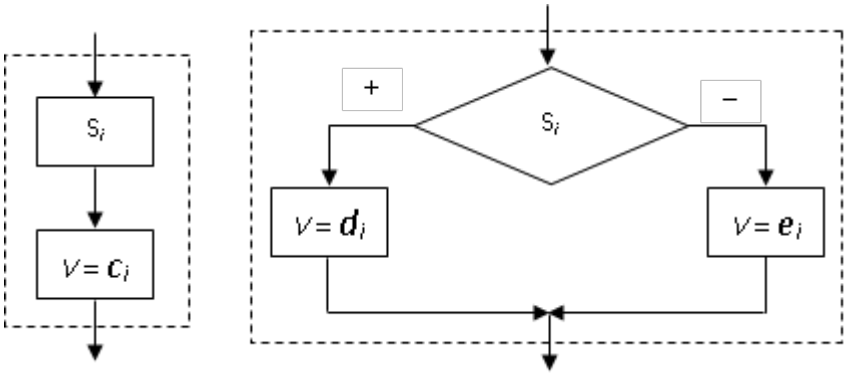


Рис. 7. Конструкція блоків еквівалентної блок–схеми

має місце твердження *T(m)*. Покажемо, що тоді має місце і твердження *T(m+1)*.

Якщо в дереві виконання блок–схеми *S* всі виконання завершені, то згідно припущення такими ж є всі виконання структурованої блок–схеми. Розглянемо довільне незавершене виконання; нехай воно закінчується блоком S_i та стрілкою, яка веде в блок S_k. Згідно побудови відповідне виконання структурованої блок–схеми закінчується блоком присвоювання *V=k*, перевіркою ряду умов та стрілкою, яка веде в блок S_k.

Вибране виконання далі продовжується блоком S_k. В залежності від типу блоку S_k це виконання може закінчуватися стрілкою (у випадку простого блоку) або розгалужуватися на два (у випадку блоку розгалуження), які теж закінчуються стрілками. Відповідне виконання структурованої блок–схеми теж продовжується блоком S_k (за яким іде присвоювання та перевірка умов) і закінчується стрілкою або розгалужується на два, які в свою чергу мають присвоювання та перевірки умов) і закінчуються стрілками. Після освітлення дерева ці шляхи (згідно побудові) – співпадуть. Таким чином, дерево виконання звичайної блок–схеми висоти *m*+1 знову співпаде з освітленим деревом виконання структурованої блок–схеми.

Побудована таким чином структурована блок–схема за рядом метрик має вищу структурну та обчислювальну складність, однак цей недолік багатократно компенсується ієрархічністю структури [5]. Саме ієрархічність структурованих програм дозволяє застосовувати до них метод програмування зверху вниз та інші сучасні технології [2].

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Практичні аспекти застосування структурованих блок–схем авторами розглянуті в роботі [6]. Тут мова структурованих блок–схем розширена композиціями неповного розгалуження, перемикача та повторення з післяумовою, що робить її більш адекватною сучасним мовам програмування. Наведено шаблони для реалізації структурованих блок–схем сучасними мовами програмування – Pascal, C++, Visual Basic.

Розширення мови структурованих блок–схем новими композиціями, очевидно, не порушує теореми представлення. Але композиція перемикача дозволяє суттєво спростити реалізацію тіла структурованого еквіваленту блок–схеми (рис. 6): один перемикач заміщає *n* композицій розгалуження.

Проблематика блок–схем та їх відображення в програмах на сучасних мовах програмування залишається в полі зору дослідників [8–10].

Наведена нотація протягом декількох років застосовується на заняттях з програмування мовами Pascal, C. C++, VBA, Java, Python, JavaScript здобувачами вищої освіти з галузі знань 12 «Інформаційні технології» і показала дуже хороші результати. Залишається актуальною задача створення графічного середовища для побудови структурованих блок–схем.

REFERENCES

[1] Knuth D. (1997) The Art of Computer Programming, vol. 1: Fundamental Algorithms. Addison-Wesley. 672. [in English].

[2] Linger R.C., Mills H.D., Witt B.I. (1979). Structured programming: theory and practice. Addison-Wesley. 402 p. [in English].

[3] Knuth, D.E., Floyd, R.W. (1971). Notes on avoiding “goto” statements. Information Processing Letters 1, Amsterdam: North-Holland. 23–31. [in English].

[4] Ashcroft E., Manna Z. (1971). The translation of “Goto” programs into “While” programs. ACM Digital Library: Proceedings of 1971 IFIP congress. P. 49–61. [in English].

[5] Yourdon E. (1979). Techniques of program structure and design. Prentice-Hall. 364 p. [in English].

[6] Kostenko A.V., Kostyrko V.S. (2010). Informatyka ta kompiuterna tekhnika. Formalizatsiia ta alhorytmizatsiia obchysliuvalnykh protsesiv: navchalnyj posibnyk [Informatics and computer technology. Formalization and algorithmization of computing processes. Tutorial]. Lviv: vydavnyctvo LKA. 200 s. [in Ukrainian].

[7] StarUML documentation. Flowchart Diagram. URL: https://docs.staruml.io/v/master/working-with-additional-diagrams/flowchart-diagram. [in English].

[8] Misra, J. (2023). Effective Theories in Programming Practice. New York: Association for Computing Machinery. 562 p. [in English].

[9] Hecht, M.S. (1977). Flow Analysts of Computer Programs. New York: North Holland. 266 p. [in English].

[10] Venit S., Drake E. (2011). Prelude to programming : concepts and design. Boston: Addison-Wesley. 630 p. [in English].

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] Knuth D. (1997) The Art of Computer Programming, vol. 1: Fundamental Algorithms. Addison-Wesley. 672.

[2] Linger R.C., Mills H.D., Witt B.I. (1979). Structured programming: theory and practice. Addison-Wesley. 402 p.

[3] Knuth, D.E., Floyd, R.W. (1971). Notes on avoiding "goto" statements. Information Processing Letters 1, Amsterdam: North-Holland. 23–31.

[4] Ashcroft E., Manna Z. (1971). The translation of "Goto" programs into "While" programs. ACM Digital Library: Proceedings of 1971 IFIP congress. P. 49–61.

[5] Yourdon E. (1979). Techniques of program structure and design. Prentice-Hall. 364 p.

[6] Костенко А.В., Костирко В.С. (2010). Інформатика та комп'ютерна техніка. Формалізація та алгоритмізація обчислювальних процесів: навчальний посібник. Львів: видавництво ЛКА. 200 с.

[7] StarUML documentation. Flowchart Diagram. URL: https://docs.staruml.io/v/master/working-with-additional-diagrams/flowchart-diagram.

[8] Misra, J. (2023). Effective Theories in Programming Practice. New York: Association for Computing Machinery. 562 p.

[9] Hecht, M.S. (1977). Flow Analysts of Computer Programs. New York: North Holland. 266 p.

[10] Venit S., Drake E. (2011). Prelude to programming : concepts and design. Boston: Addison-Wesley. 630 p.

Дата надходження статті до редакції: 23.03.2023
Дата затвердження статті до друку: 30.04.2023