МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Ю.В. Ришковець, В.А. Висоцька

А Л Г О Р И Т М І З А Ц І Я Т А П Р О Г Р А М У В А Н Н Я

Частина 2

Навчальний посібник

Видавництво «Новий Світ-2000»

Львів

2021

УДК 81:004.93

Гриф надано Науково-методичною радою
Національного університету «Львівська політехніка»
(Протокол № 33 від «13» грудня 2017 р.)

Рецензенти:
Гожий О.П. – доктор технічних наук, професор кафедри комп'ютерної інженерії Чорноморського національного університету імені Петра Могили;
Голощук Р.О. – кандидат технічних наук, доцент кафедри соціальних комунікацій та інформаційної діяльності Інституту гуманітарних та соціальних наук Національного університету «Львівська політехніка»;
Литвиненко В.І. – доктор технічних наук, професор, завідувач кафедри інформатики і комп'ютерних наук Херсонського технічного університету;
Марікуца У.Б. – кандидат технічних наук, доцент кафедри систем автоматизованого проектування, декан базової вищої освіти, заступник директора інституту комп'ютерних наук та інформаційних технологій Національного університету «Львівська політехніка»;
Бісікало О.В. – доктор технічних наук, професор, завідувач кафедри автоматики та інформаційно-вимірювальної техніки Вінницького національного технічного університету;
Шаховська Н.Б. – доктор технічних наук, професор, завідувач кафедри систем штучного інтелекту Національного університету «Львівська політехніка».

Ришковець Ю.В., Висоцька В.А.

Алгоритмізація та програмування. Частина 2: навчальний посібник – Львів: Видавництво «Новий Світ-2000», 2021. – 315 с.

ISBN 978–617–7519–17–0

Навчальний посібник містить матеріал для вивчення основних теоретичних засад, функціональних можливостей та практичного застосування теорії алгоритмів та основ програмування, розроблення прикладних засобів та інформаційних систем аналізу та опрацювання інформації за допомогою алгоритмів. Теоретичний та практичний матеріал викладено у доступній формі. Викладення матеріалу супроводжується значною кількістю прикладів, що полегшує його сприйняття та засвоєння. Подається перелік питань та тестів для самоконтролю, а також завдань для самостійного виконання трьох рівнів складності. Навчальний посібник призначається для студентів, що навчаються за спеціальностями 122 «Комп'ютерні науки», 124 «Системний аналіз», 126 «Інформаційні системи та технології» та споріднених спеціальностей, які пов'язані з інформатикою та інформаційними технологіями. Він може бути використаний аспірантами як підґрунтя для наукових досліджень та викладачами як дидактичний матеріал, а також для самостійного вивчення. Книга призначена для спеціалістів із проектування, розроблення та впровадження інтелектуальних систем опрацювання інформаційних ресурсів, науковців у галузі глобальних інформаційних системи, систем штучного інтелекту, Інтернет-технологій, фахівців з електронної комерції, Інтернет-маркетингу та Інтернет-реклами, менеджерів комплексних Web-проектів, а також для здобувачів 3-ого (освітньо-наукового) рівня вищої освіти в галузі знань 12 «Інформаційні технології». Кожний розділ закінчується переліком питань для самоконтролю, прикладом тестових питань з відповідями та переліком індивідуальних завдань для виконання лабораторних робіт.

ISBN 978–617–7519–17–0

ЗМІСТ

Вступ

Алгоритм – це система формальних правил, які чітко та однозначно визначають послідовність дій обчислювального процесу від початкових даних до шуканого результату. Слово алгоритм походить від algorithm – латинської форми написання імені великого математика IX ст. Аль-Хорезмі, який сформулював правила виконання арифметичних дій. Таким чином алгоритм визначає певні правила перетворювання інформації, тобто визначає певну послідовність операцій опрацювання даних для отримання розв'язку задачі. Алгоритм – це скінченна послідовність команд, які треба виконати над вхідними даними для отримання результату. Алгоритм розгляда.nm як інструмент, який призначений для вирішення коректно поставленої обчислювальної задачі. У постановці задачі в загальних рисах визначаються відношення між входом та виходом. В алгоритмі описується конкретна обчислювальна процедура, за допомого якої можна досягнути виконання вказаних відношень. До основних ознак алгоритму належать такі:

- Дискретність інформації. Кожний алгоритм має справу з даними: вхідними, проміжними, вихідними. Ці дані подають у вигляді скінченних слів деякого алфавіту.
- Дискретність роботи алгоритму. Алгоритм виконується по кроках та при цьому на кожному кроці виконується тільки одна операція.
- Детермінованість алгоритму. Система величин, які отримуються в кожний (не початковий) момент часу, однозначно визначається системою величини, які були отримані в попередні моменти часу.
- Елементарність кроків алгоритму. Закон отримання наступної системи величин з попередньої повинен бути простим та локальним.
- Виконуваність операцій. В алгоритмі не має бути не виконуваних операцій. Наприклад, неможна в програмі призначити значення змінній «нескінченність», така операція була би не виконуваною. Кожна операція опрацьовує певну ділянку слова.
- Скінченність алгоритму. Опис алгоритму повинен бути скінченним.
- Спрямованість алгоритму. Якщо спосіб отримання наступної величини з деякої заданої величини не дає результату, то має бути вказано, що треба вважати результатом алгоритму.
- Масовість алгоритму. Початкова система величин може обиратись з деякої потенційно нескінченної множини.

Перелічимо переваги використання алгоритмізації в програмуванні.

По-перше, алгоритми є життєво необхідними складовими для рішення будь-яких задач з різноманітних напрямків комп'ютерних наук. Алгоритми відіграють ключову роль у сучасному розвитку технологій. Тут можна згадати такі розповсюджені задачі, як:

- розв'язання математичних рівнянь різної складності, знаходження добутку матриць, обернених матриць;
- знаходження оптимальних шляхів транспортування товарів та людей;

- знаходження оптимальних варіантів розподілення ресурсів між різними вузлами (виробниками, верстатами, працівниками, процесорами тощо);
- знаходження в геномі послідовностей, які співпадають;
- пошук інформації в глобальній мережі Інтернет;
- прийняття фінансових рішень в електронній комерції;
- опрацювання та аналіз аудіо та відео інформації.

Порівняно з програмою, алгоритм може перебувати на вищому рівні абстракції, бути вільним від деталей реалізації, пов'язаних з особливостями мови програмування та конкретної обчислювальної системи. Засоби, прийняті для зображення алгоритмів, переважно називають алгоритмічною мовою.

Для подання алгоритма використовують такі способи:

- словесний опис послідовності дій, який потребує подальшої формалізації;
- аналітичний опис у вигляді таблиць, формул, схем, малюнків тощо дає змогу описати алгоритм за допомогою системи умовних позначень і є найбільш формалізованим;
- графічне подання у вигляді схеми алгоритму (блок-схеми) використовується для наочності подання алгоритму за допомогою набору спеціальних блоків;
- запис алгоритмічною мовою програмування; алгоритм записується зрозумілою для комп'ютера мовою, тобто мовою програмування.

Схема алгоритму (блок-схема) – це графічне зображення його структури, в якому кожний етап процесу опрацювання даних подається у вигляді певних геометричних фігур (блоків). Форма блока визначає спосіб дій, а записи всередині блока – деталі (змінні, параметри) відповідного етапу.

Усередині блоків слід розміщувати мінімальну кількість тексту, необхідну для розуміння того, що виконує певний блок. Текст для читання записується зліва направо і зверху вниз незалежно від напряму лінії потоку.

Якщо обсяг тексту, розміщеного всередині блока, перевищує його розміри, то використовується коментар.

Напрям лінії потоку зліва направо і зверху вниз є стандартним. У випадках, коли потрібно внести певну ясність у схему, наприклад, при з'єднаннях, на лініях використовують стрілки. Якщо напрям потоку нестандартний, то стрілки мають відображати цей напрям.

Перевагою блок-схем є те, що за допомогою них можна наочно зобразити структуру алгоритму цілковито, окресливши його логічну сутність.

Розрізняють три базових алгоритмічних структури – лінійна (послідовність), розгалужена та циклічна (повторювання). Алгоритмічні структури будь-якої складності утворюються комбінацією послідовності базових структур.

Запис алгоритму алгоритмічною мовою програмування має бути зрозумілим не лише людині, а й комп'ютеру і тому потрібно точно дотримуватися правил цієї мови [59].

До основних властивостей алгоритмів належать такі:

- визначеність (детермінованість) – однозначність результату обчислювального процесу при заданих початкових даних;

- дискретність (скінченність) – поділ обчислювального процесу на окремі елементарні кроки, що виконуються послідовно;
- ефективність – простота та швидкість розв'язання задачі при використанні мінімально необхідних засобів;
- правильність – забезпечення отримання розв'язку;
- результативність – забезпечення отримання розв'язку через певну скінченну кількість кроків;
- масовість – забезпечення розв'язку будь-якої однотипної задачі [59].

Алгоритм визначений, якщо він складається з допустимих команд виконавця, які можна виконати для деяких вхідних даних. Невизначеність виникне в алгоритмі (x+y)/(*a*-*b*), якщо в знаменнику буде записано, наприклад, 92 – 92 (ділення на нуль неприпустиме). Невизначеність виникне, якщо деяка команда буде записана неправильно, бо така команда не належатиме до набору допустимих команд виконавця, (*x*+*y*)/(*a*-*b*).

Алгоритм повинен бути дискретним – послідовність команд, які треба виконати, має бути скінченною. Кожна команда починає виконуватися після закінчення виконання попередньої.

Алгоритм (*x*+*y*)/(*a*-*b*) – скінченний. Він складається з трьох дій. Кожна дія, у свою чергу, реалізується скінченною кількістю елементарних арифметичних операцій. Нескінченну кількість дій передбачає математичне правило перетворення деяких звичайних дробів, таких як 5/3, у нескінченні десяткові дроби.

Алгоритм результативний, якщо він дає результати, які можуть виявитися і неправильними. Наведені вище алгоритми є результативними. Прикладом нерезультативного алгоритму буде алгоритм для виконання обчислень, в якому пропущена команда виведення результатів на екран тощо. Алгоритм правильний, якщо його виконання забезпечує досягнення мети.

Алгоритм формальний, якщо його можуть виконати не один, а декілька виконавців з однаковими результатами. Ця властивість означає, що коли алгоритм *A* застосовують до двох однакових наборів вхідних даних, то й результати мають бути однакові.

Алгоритм масовий, якщо він придатний для розв'язування не однієї задачі, а багатьох задач певного класу. Прикладами масових алгоритмів є загальні правила, якими користуються для множення, додавання, ділення двох багатозначних чисел, бо вони застосовні для будь-яких пар чисел. Масовими є алгоритми розв'язування математичних задач, описаних у загальному вигляді за допомогою формул, їх можна виконати для різних вхідних даних.

Основними мірами обчислювальної складності алгоритмів є:

- часова складність, яка характеризує час, необхідний для виконання алгоритму на певній машині; цей час, як правило, визначається кількістю операцій, які треба виконати для реалізації алгоритму;
- ємнісна складність, яка характеризує обсяг пам'яті, необхідний для виконання алгоритму.

Часова та ємнісна складність тісно пов'язані між собою. Обидві є функціями від розміру вхідних даних. Надалі обмежимося тільки аналізом часової складності.

Складність алгоритму описується функцією $f(n)$, де n – розмір вхідних даних. Важливе теоретичне і практичне значення має класифікація алгоритмів, яка бере до увагу швидкість зростання цієї функції.

Основною оцінкою функції складності алгоритму $f(n)$ є оцінка Θ.

Кажуть, що $f(n) = \Theta(g(n))$, якщо при $g > 0$ при $n > 0$ існують додатні c_1, c_2, n_0, такі, що $c_1 g(n) \le f(n) \le c_2 g(n)$ при $n > n_0$. Тобто, можна знайти такі c_1 та c_2, що при достатньо великих n функція $f(n)$ знаходитиметься між $c_1 g(n)$ та $c_2 g(n)$.

У такому випадку функція $g(n)$ є асимптотично точною оцінкою функції $f(n)$, оскільки за визначенням функція $f(n)$ не відрізняється від функції $g(n)$ з точністю до постійного множника. Виділяють такі основні класи алгоритмів:

- *логарифмічні*: $f(n) = \Theta(\log_2 n)$;
- *лінійні*: $f(n) = \Theta(n)$; якщо n=1, то отримуємо константні алгоритми;
- *поліноміальні*: $f(n) = \Theta(n^m)$; тут m – натуральне число, більше від одиниці; при $m = 1$ алгоритм є лінійним;
- *експоненційні*: $f(n) = \Theta(a^n)$; a – натуральне число, більше від одиниці. Експоненційні алгоритми часто пов'язані з перебором різних варіантів розв'язку.

Для однієї й тої ж задачі можуть існувати алгоритми різної складності. Часто буває й так, що повільніший алгоритм працює завжди, а швидший – лише за певних умов. Будемо називати часовою складністю задачі часову складність найефективнішого алгоритму для її розв'язання. Алгоритми без циклів і рекурсивних викликів мають константну складність. Якщо немає рекурсії та циклів, всі керуючі структури можуть бути зведені до структур константної складності. Отже, і весь алгоритм також характеризується константною складністю. Визначення складності алгоритму в основному зводиться до аналізу циклів і рекурсивних викликів. Але, якщо заздалегідь відомо, що послідовність упорядкована за зростанням або за спаданням, можна застосувати інший алгоритм – алгоритм половинного ділення. Послідовність ділиться на дві рівні частини. Оскільки послідовність упорядкована, можна визначити, в якій частині міститься потрібний елемент. Після цього процедура повторюється: потрібна частина знову ділиться навпіл і т.д. Цей алгоритм є логарифмічним.

Експоненційні алгоритми часто пов'язані з перебором різних варіантів розв'язання. Наведемо типовий приклад. Типова схема розв'язування цієї задачі може мати такий вигляд: спочатку надаємо одне з двох можливих значень (0 або 1) першій змінній x_1, потім другій і т.ін. Коли будуть розставлені значення всіх змінних, можна визначити значення булевого виразу. Якщо він дорівнює 1, задача розв'язана. Якщо ні – треба повернутися назад і змінити значення деяких змінних. Можна інтерпретувати цю задачу як задачу пошуку на певному дереві перебору. Кожна вершина цього дерева відповідає певному набору встановлених значень $x_1 \ldots, x_k$. Можна присвоїти змінній x_{k+1} значення 0 або 1,

тобто маємо вибір з двох дій. Тоді кожній із цих дій відповідає одна з двох дуг, які йдуть від цієї вершини. Вершина $x_1 \ldots, x_k$ має двох синів $x_1 \ldots, x_k 0$ і $x_1 \ldots, x_k 1$. Звертаємо увагу на те, що не кожен перебірний алгоритм є експоненційним (наприклад, алгоритм пошуку в масиві, незважаючи на його перебірний характер, є лінійним, а не експоненційним). Зі зростанням розмірності будь-який поліноміальний алгоритм стає ефективнішим за будь-який експоненційний. Для лінійного алгоритму зростання швидкодії комп'ютера в 10 разів дає змогу за той же час розв'язати задачу, розмір якої в 10 разів більший. Для експоненційного алгоритму з основою 2 цей же розмір можна збільшити лише на 3 одиниці. Як правило, якщо для розв'язування певної задачі існує певний поліноміальний алгоритм, то часова оцінка цього алгоритму значно покращується.

Метод перебору із поверненнями дає змогу розв'язувати практично незліченну множину задач, для багатьох з яких невідомі інші алгоритми. Незважаючи на таке велике різноманіття перебірних задач, в основі їх розв'язування є щось спільне, що дає змогу застосувати цей метод. Таким чином, перебір можна вважати практично універсальним методом розв'язування перебірних завдань. Наведемо загальну схему цього методу. Розв'язування задачі методом перебору з поверненням будується конструктивно послідовним розширенням часткового розв'язування. Якщо на конкретному кроці таке розширення провести не вдається, то відбувається повернення до коротшого часткового розв'язування, і спроби його розширити продовжуються.

Машина Тьюринґа, що була описана А.Тьюринґом 1936 року, є теоретичною моделлю обчислювальної машини. Машину Тьюринґа (МТ) (рис. 1.5) слід розглядати як одну з можливих формалізацій поняття алгоритму. Її робота може бути описана таким чином. Розглянемо стрічку, розділену на окремі комірки; ця стрічка є потенційно нескінченною в обидва боки. У кожній комірці може бути записаний певний символ з деякого заданого алфавіту А. Машина Тьюринґа в будь-який момент часу може перебувати в певному стані (множина станів S є скінченною) і вказувати на певну комірку.

Машина Тьюринґа в залежності від поточного символа, на який вона вказує, може записати на його місце будь-який інший символ (він може співпадати зі старим), зсунутися на один символ вліво або вправо, змінити свій вміст чи зупинитися (часто вважається, що машина зупиняється автоматично, якщо немає жодної інструкції, яку вона могла б виконати). Робота машини Тьюринґа визначається її програмою. Програма машини Тьюринґа є послідовністю інструкцій, кожна з яких має вигляд

$$a_i s_j \rightarrow a_k s_l I, \text{ де } a_i, a_k \in A; s_l, s_m S, I \in \{R, L, H\}.$$

Цей запис читається так: якщо машина перебуває в стані s_l і зчитує символ a_i, вона повинна записати в поточну позицію символ a_k, перейти до стану s_m і зсунутися вправо (відповідає літері R), вліво (відповідає літері L) або зупинитися (відповідає літері H). На початку роботи машина перебуває на лівому кінці стрічки в початковому стані s_0. Вона виконує операції, що визначаються її програмою. Якщо вона в деякий момент зупиняється,

результатом роботи алгоритму вважається послідовність символів, яка записана на стрічці в момент зупинки.

Універсальну машину Тьюринґа неформально визначають як машину, що може сприймати програму для обчислення будь-якої функції, яку, в принципі, можна обчислити за допомогою спеціалізованої машини М1 і надалі працювати, як машина М1. Також можна довести, що таку машину можна побудувати. Різноманіття можливостей конструкції Тьюринґа полягають у тому, що якщо певні алгоритми А та В реалізуються машинами Тьюринґа, то можна будувати програми машин Тьюринґа, які реалізують композиції алгоритмів А та В. Наприклад, виконати А, потім виконати В або виконати А знову. Якщо в результаті утворилося слово “так”, то виконати В. У протилежному випадку не виконувати В або виконувати по черзі А, В, поки В не дасть відповідь “ні”. В інтуїтивному сенсі такі композиції є алгоритмами. Тому їхня реалізація за допомогою машини Тьюринґа служить одним із засобів обґрунтування універсальності конструкції Тьюринґа. Реалізованість таких композицій доводиться у загальному вигляді, незалежно від особливостей конкретних алгоритмів А та В. Доведення полягає в тому, що вказується засіб побудови з програм А та В програми необхідної композиції. Аналогічно конструюють й інші композиції машин Тьюринґа; щораз будуються загальні правила, які визначають, що на що змінювати у вихідних програмах. Описуючи різноманітні алгоритми для машин Тьюринґа і стверджуючи реалізованість усіляких композицій алгоритмів, Тьюринґ переконливо показав розмаїтість можливостей запропонованої ним конструкції, що дозволило йому виступити з такою тезою: *будь-який алгоритм може бути реалізований відповідною машиною Тьюринґа*. Це основна гіпотеза теорії алгоритмів у формі Тьюринґа. Одночасно ця теза є формальним визначенням алгоритму. Завдяки їй можна доводити існування або неіснування алгоритмів, створюючи відповідні машини Тьюринґа або доводячи неможливість їхньої побудови. Завдяки цьому з’являється загальний підхід до пошуку алгоритмічних розв’язків.

Якщо пошук розв’язку наштовхується на перешкоду, то можна використовувати цю перешкоду для доведення неможливості розв’язування, спираючись на основну гіпотезу теорії алгоритмів. Якщо ж при доказі неможливості виникає своя перешкода, то вона може допомогти просунутися в пошуку розв’язку, хоча б частково усунувши стару перешкоду. Так, по черзі намагаючись довести то існування, то відсутність розв’язку, можна поступово наблизитися до розуміння суті поставленої задачі. Довести тезу Тьюринґа не можна, тому що в його формулюванні не визначене поняття будь-який алгоритм, тобто ліва частина тотожності. Його можна тільки обґрунтувати, подаючи різноманітні відомі алгоритми у вигляді машин Тьюринґа. Додаткове обґрунтування цієї тези полягає в тому, що пізніше було запропоновано ще декілька загальних визначень поняття алгоритму і щораз вдавалося довести, що, хоча нові алгоритмічні схеми і виглядають інакше, вони насправді еквівалентні машинам Тьюринґа: *усе, що реалізовано в одній з цих конструкцій, можна зробити і в інших*. Ці твердження доводяться строго, тому що в них мова йде вже про тотожність формальних схем.

Теза Чорча часто формулюється в еквівалентній формі, а саме: будь-який алгоритм в інтуїтивному розумінні цього слова може бути реалізований за допомогою деякої машини Тьюринґа. Іншими словами, за допомогою машини Тьюринґа можна розв'язати будь-яку задачу, для якої існує алгоритм розв'язування в інтуїтивному розумінні. Довести тезу Чорча неможливо. Її можна було б спростувати, якби були запропоновані формалізації поняття алгоритму, здатні обчислювати нерекурсивні функції. Але таких формалізмів досі запропоновано не було. Якщо визнати тезу Чорча, то можна прийняти машину Тьюринґа як основу для загального визначення алгоритму: алгоритм є послідовністю інструкцій, яка може бути виконана за допомогою машини Тьюринґа або еквівалентної їй обчислювальної моделі. Тепер можна дати більш чітке визначення універсального комп'ютера: універсальним називається комп'ютер, за допомогою якого можна промоделювати роботу машини Тьюринґа. Якщо теза Чорча є справедливою, ми маємо визнати наступне. Якщо вдається довести, що не існує машини Тьюринґа, яка могла б вирішити певну проблему, то ця проблема є алгоритмічно нерозв'язною, тобто для неї не існує загального алгоритму розв'язування. Такі проблеми насправді існують. Наприклад, це знаменита проблема зупинки, яка формулюється так: потрібно для будь-яких початкових даних визначити, зупиниться машина Тьюринґа чи ні. Оскільки будь-яка програма, яка працює на будь-якому комп'ютері, може бути реалізована і на машині Тьюринґа, це твердження можна переформулювати так: не існує загального алгоритму, який для будь-якої програми заздалегідь визначав би, зупиниться вона чи ні. Але можна навести формалізми, які полегшують програмування і дають змогу здійснювати обчислення швидше, ніж машини Тьюринґа.

Навчальний посібник містить матеріал для вивчення основних теоретичних засад, функціональних можливостей та практичного застосування теорії алгоритмів та основ програмування, розроблення прикладних засобів та інформаційних систем аналізу та опрацювання інформації за допомогою алгоритмів. Викладення матеріалу супроводжується значною кількістю прикладів, що полегшує його сприйняття та засвоєння. Навчальний посібник призначається для студентів, що навчаються за спеціальностями 122 «Комп'ютерні науки», 124 «Системний аналіз» та 126 «Інформаційні системи та технології» і споріднених спеціальностей, які пов'язані з вивченням чисельних методів в інформатиці та інформаційних технологій. Може бути використаний аспірантами як підґрунтя для наукових досліджень та викладачами як дидактичний матеріал, а також для самостійного вивчення та підвищення кваліфікації. Книга призначена для спеціалістів із проектування, розроблення та впровадження інтелектуальних систем опрацювання інформаційних ресурсів, науковців у галузі глобальних інформаційних системи, систем штучного інтелекту, Інтернет-технологій, фахівців з електронної комерції, Інтернет-маркетингу та Інтернет-реклами, менеджерів комплексних Web-проектів, а також для здобувачів 3-го (освітньо-наукового) рівня вищої освіти в галузі знань 12 «Інформаційні технології». Кожний розділ закінчується переліком питань для самоконтролю, прикладом тестових питань з відповідями

до екзаменаційних білетів та переліком індивідуальних завдань для виконання лабораторних робіт.

Навчальний посібник складається з двої частина Перша чатисна присвячена програмуванню на мові С. Розділ 1 першої частини присвячений елементам обчислювальних машин та описує архітектуру комп'ютерів. Детально пояснені архітектурні принципи Джона фон Неймана та позиційні системи числення (двійкова, вісімкова, шістнадцяткова система числення, та взаємозв'язок між ними). Розкрито поняття алгоритму та способи його подання. Описані машинні та високорівневі мови програмування. Розділ 2 знайомить з історією розвитку мови С та з основними засадами програмування, зокрема такими як алфавіт мови С, структура програми, типи даних, змінні та константи. Розділ 3 присвячений основним операціям в мові С (операції присвоєння, арифметичним операціям, операціям перетворення типів, операціям відношень та логічним операціям). Також тут розглядається введення і виведення даних. У розділі 4 розглянуто поняття розгалуження, описані оператори безумовного та умовного переходів, а також оператор вибору варіантів. Поняття ітерації та циклу описані в розділі 5. Зокрема тут розглядаються цикли з параметром, передумовою, післяумовою та оператори переривання циклів. Вказівники та посилання подані у розділі 6.

Розділ 7 присвячений створенню підпрограм, в якому описано поняття підпрограми, параметри та їх види, статичні змінні, передавання параметрів функцій (параметри-значення, параметри-вказівники, параметри-посилання, параметри зі значеннями за замовчуванням, імена функцій як параметри), вбудовані функції та функції зі змінною кількістю параметрів. У розділі 8 визначено поняття рекурсії та рекурентних співвідношень. Розділ 9 присвячений модульному програмуванню. Тут описано основні поняття модульного програмування, використання користувацьких файлів, проблема подвійного включення, та умовна компіляція. Розділ 10 знайомить із поняттям масиву. Розглянуто одновимірні масиви (оголошення одновимірних масивів, операції над вказівниками на масиви, введення-виведення одновимірних масивів) і багатовимірні масиви (двовимірні та тривимірні масиви), а також опрацювання масивів у функціях. В розділі 11 описані загальні поняття динамічної пам'яті, функції для роботи з динамічною пам'яттю, динамічні одновимірні і двовимірні масиви та їх опрацювання у функціях. Розділ 12 описує особливості роботи із символьними рядками. Детально описано поняття символьного типу (коди символів, опрацювання символів, введення та виведення символів), а також рядків символів (оголошення рядків, введення-виведення рядків та їх опрацювання). Розглядається опрацювання символьних рядків у функціях. Розділ 13 присвячений комбінованим типам, зокрема, структурам (визначення структурного типу, його оголошення та операції над ними), опрацюванню структур у функціях та бітовим полям. Файли детально описані в розділі 14, а саме загальні поняття, потоки та файли, додаткові можливості опрацювання файлів, текстові та бінарні файли.

Друга частина навчального посібника акцентує увагу на алгоритміації. Розділ 1 другої частини присвячений динамічним структурам даних та

алгоритмам їх опрацювання, тобто спискам, чергам, стекам та бінарним деревам. У розділі 2 описана загальна класифікація алгоритмів пошуку та основні відомі алгоритми пошуку такі як лінійний, двійковий (бінарний) пошук елемента в масиві, методом Фібоначчі, інтерполяційний пошук, бінарний пошук із визначенням найближчих вузлів. Розділ 3 містить опис основних алгоритмів сортування – сортування обміном (метод бульбашки), вставкою (включенням), прямим вибором та швидке (метод Хоара) сортування. Приділена увага сортуванню включенням зі спадним приростом (метод Шелла), сортуванню за допомогою дерева (пірамідальне сортування) та сортуванню злиттям. Розділ 4 присвячений нелінійним структурам даних та алгоритмам їх опрацювання.

Автори цього навчального посібника висловлюють щиру подяку фахівцям в галузі знань 12 "Інформаційні технології" за плідну співпрацю, вагомі поради та слушні зауваження при формуванні структури книги та її наповнення. Зокрема, дякуємо за підтримку:

Гожому О.П. – доктору технічних наук, професору кафедри комп'ютерної інженерії Чорноморського національного університету імені Петра Могили;

Голощуку Р.О. – кандидату технічних наук, доценту кафедри соціальних комунікацій та інформаційної діяльності Інституту гуманітарних та соціальних наук Національного університету «Львівська політехніка»;

Григоровичу В.Г. – кандидату технічних наук, доценту кафедри інформаційних систем і технологій Дрогобицького державного педагогічного університету імені Івана Франка;

Кравцю П.О. – кандидату технічних наук, доценту кафедри інформаційних систем та мереж Національного університету «Львівська політехніка»;

Литвиненку В.І. – доктору технічних наук, професору, завідувачу кафедри інформатики і комп'ютерних наук Херсонського технічного університету;

Литвину В.В. – доктору технічних наук, професору, завідувачу кафедри інформаційних систем та мереж Національного університету «Львівська політехніка»;

Марікуці У.Б. – кандидату технічних наук, доценту кафедри систем автоматизованого проектування, декану базової вищої освіти, заступник директора інституту комп'ютерних наук та інформаційних технологій Національного університету «Львівська політехніка»;

Бісікалу О.В. – доктору технічних наук, професору, завідувачу кафедри автоматики та інформаційно-вимірювальної техніки Вінницького національного технічного університету;

Шароновій Н.В. – доктору технічних наук, професору, завідувачу кафедри інтелектуальних комп'ютерних систем Національного технічного університету «ХПІ»;

Шаховській Н.Б. – доктору технічних наук, професору, завідувачу кафедри систем штучного інтелекту Національного університету «Львівська політехніка»;

Шпак З.Я. – кандидату технічних наук, доценту кафедри автоматизованих систем управління Національного університету «Львівська політехніка».