Тема 5 ПРОЕКТНА ПРОЦЕДУРА РОЗРОБКИ ФУНКЦІОНАЛЬНИХ ОПИСІВ

5.1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО ПРОЕКТНИЙ ПРОЦЕДУР

Розвиток окремих напрямів програмування, філології, психології, теорії проектування та штучного інтелекту наблизився до точки, коли відчувається нагальна необхідність інтеграції накопичених результатів. Спроба такої інтеграції втілилася у проектній процедурі (методиці), яка може бути застосована для складання функціональних описів (структурованих описів процесів). Інструкція використання будь-яким пристроєм, інструкція взагалі або алгоритм програми є описами функціонування. Описи функціонування можуть бути як структурованими відповідно до підходу до структурування, що викладається, так і неструктурованими. Таким чином, у цьому розділі розглядається саме проектна процедура (методика) розробки функціональних описів функціонування систем, що відрізняється використанням особливого структурування.

Програмісти можуть застосовувати проектну процедуру під час написання:

- Документальних описів обчислювальних та інших алгоритмів, призначених для тиражування;
- Інструкцій роботи користувача в системі організації (установи) з використанням ЕОМ та програми, що розробляється;
- описів зовнішнього функціонування програми у формі сценарію роботи програми (такі описи передують внутрішньому проектуванню програми);
- Внутрішніх специфікацій функціонування програми (методу вирішення задачі, алгоритму програми в цілому);
- Вихідних текстів модулів програми при використанні технології структурного програмування;
- код методів об'єктів при використанні технології об'єктно-орієнтованого програмування;
- Текстів допомоги з використання програми.

Вміння застосовувати проектну процедуру є корисним і непрограмістам. За допомогою цієї проектної процедури можна складати:

• короткі та зрозумілі описи будь-яких процесів;
• накази та розпорядження на виконання робіт;
• інструкції щодо користування виробами;
• опис принципів функціонування виробів;
• опис бізнес-процесів;
• бухгалтерські інструкції щодо проведення розрахунків;
• тексти посадових вказівок організаційного забезпечення.

Цей список не є вичерпним і можливі нові застосування.

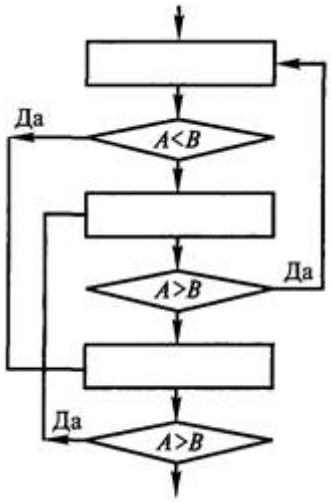
Докладніше актуальність розробки функціональних описів поза сферою програмування характеризується такими прикладами.

«Копати траншею від паркану до обіду» — невдале розпорядження (недостатньо повно виявлено та вказано вхідну та вихідну інформацію, а також відсутня мета).
Припустимо, що наприкінці червня якийсь вітчизняний бухгалтер звернувся із проханням про відпустку. Головний бухгалтер, найімовірніше, дасть приблизно таку відповідь: "Не дам жодної відпустки до здачі піврічного звіту". Уявімо, що є набір щоденних детально вичерпних інструкцій по роботі даного бухгалтера. Набір таких інструкцій називається описом бізнес-процесів. І тут і у разі хвороби робота бухгалтера передається резервному спеціалісту. Припустимо, що зверху спущений якийсь документ, згідно з яким треба подати якісь дані, які раніше не розраховувалися. Швидше за все, цей документ є заплутаною інструкцією, що допускає різночитання або навіть містить помилки для застосування в деяких особливих випадках. Джерелом таких помилок може бути не злий намір укладача інструкції, а нездатність її упорядника охопити всі особливі випадки з величезного (астрономічного!) кількості шляхів виконання інструкції, розробленої за традиційною операційною методикою. Тепер головний бухгалтер починає обдзвонювати колег та інстанції, сподіваючись отримати роз'яснення. Нехай він розібрався в інструкції. Тепер перед ним постає завдання по віддачі розпоряджень працівникам бухгалтерії як працювати за умов, що змінилися. Багато бухгалтерів, не знаючи основ сучасної алгоритмізації, швидше за все, звернуться до програмістів, які добре оплачуються, для уникнення труднощів у віддачі розпоряджень.
На відміну від бухгалтера, який документально фіксує всі свої дії, виконроб на будівництві свої розпорядження з виробництва робіт готує та віддає усно. Це пояснюється традиційним ухиленням від відповідальності у разі травматизму робітників. Усна підготовка розпоряджень призводить до таких помилок, як щось не вдається розмістити у вже збудованій частині будівлі. Таким чином, будівництво ведеться циклічним процесом: будувати – ламати – будувати тощо, що веде до подорожчання будівництва.
Продаж і навіть перепродаж готових технічних виробів за чинним законодавством передбачає наявність інструкції користування виробом російською мовою. Однак часто виявляється, що інженери складають такі довгі та заплутані інструкції користування об'єктами техніки, що споживачі починають експлуатацію виробу з неприпустимих дій, які можуть призвести до виробу навіть до неремонтно-придатного стану.
Крім апробації у сфері програмування автори підручника провели апробацію викладених у ньому методик під час навчання непрограмуючих спеціальностей. Виявилося, що навчання методики розробки функціональних описів (на прикладах складання інструкцій взагалі, описів бізнес-процесів) цілком доступне студентам другого курсу спеціальності бухгалтерський облік, навіть якщо вони не вивчали цю методику в курсі програмування. Понад те, половина учнів дев'ятого класу звичайної школи цілком здатна повністю освоїти цей матеріал. Слід зазначити такий факт: до навчання лише кілька учнів класу реально могли написати план, та був твір. Після навчання майже три чверті учнів могли написати план, та був його розвинути у твір, т. е. школярі реально освоїли елементи дедуктивного мислення! Витрати освоєння матеріалу склали 8 год лекційних і 16 год практичних занять. Таким чином, у навчальних всього за 24 години навчальних занять вдається розвинути первинні навички дедуктивного мислення і володіння початковими методами системного підходу.
Написання технічної документації – особливий жанр письменницького мистецтва. Нині у розвинених країнах з'являється нова спеціальність Technical Writer – технічний письменник. Ймовірно, одна із сфер застосування проектної процедури полягає у її використанні такими фахівцями.
Хорошим функціональним описом є безпомилковий опис, однозначний для читача, короткий, суть якого розуміється швидко. Згідно з проектною процедурою, хороший функціональний опис складається від загального до приватного з використанням особливих конструкцій пропозицій — типових елементів (типових структур або просто структур), що становлять семантичний скелет майбутніх інструкцій. Зазвичай людина мислить пропозиціями природної мови. Якщо навчитися впорядковувати думки у процесі мислення, можна навчитися отримувати алгоритми та інші функціональні описи зі швидкістю як не меншою, ніж до навчання, і навіть більшої. Досвідчений програміст пише текст мовою програмування зі швидкістю, з якою він думає, а разі простих алгоритмів — зі швидкістю набору тексту на клавіатурі. Інструкції, призначені для виконання людьми, можуть містити алгоритми і евроритми. Одна з переваг застосування проектної процедури полягає в зниженні розумової втоми програміста або укладача інструкцій за рахунок виключення необхідності неодноразового повторення розумового процесу для отримання однієї і тієї ж ідеї, що забувається.

Головна перевага полягає в однозначності відповідності функціонального опису задуму, що досягається вичерпним тестуванням. При операційному підході до складання описів функціонування вичерпне тестування принципово неможливе через складність розв'язуваної задачі (потрібно відразу відтестувати велику і складну структуру - всю програму чи інструкцію).
Ще одна перевага полягає у отриманні само документованих текстів програм. Само документовані програми виходять шляхом застосування спеціального стандартизованого способу оформлення текстів програм з використання коментарів та стандартних типових структур кодування.
Використання стандартних типових структур передбачає особливу декомпозицію алгоритму програми чи євроритму інструкції за принципом від загального до приватного, що вимагає від розробника володіння дедуктивним мисленням.

5.2. ІСТОРІЯ ВИНИКНЕННЯ ПРОЕКТНОЇ ПРОЦЕДУРИ

З появою ЕОМ актуальним став пошук методів опису обчислювальних алгоритмів. У 60-х роках вже застосовувалися два способи опису алгоритмів: словесний покроковий та графічний у вигляді схем алгоритмів програм (жаргонно: блок-схем алгоритмів).



*Мал. 5.1.*Зображення алгоритму у формі графічної схеми алгоритму

При словесно-покроковому способі алгоритми описувалися за наведеним нижче принципом.
*Крок 1*Виконується така дія для того-то. Якщо виходить, що A <, то переходимо до кроку 4.
*Крок 2.*Виконується така дія для того-то.
*Крок 3*Якщо A > B, переходимо до виконання кроку 1.
*Крок 4.*Виконується така дія для того-то.
*Крок 5.*Якщо A > B, переходимо до виконання кроку 2.
Зображення цього алгоритму у вигляді схеми алгоритму наведено на рис. 5.1.
Недоліки кожного із способів наведені у табл. 5.1, з якої видно, що графічний спосіб у вигляді схем алгоритмів програм полегшив лише відстеження передач управління та одночасно ускладнив опис суті процесів (їх коментування).
Таблиця 5.1
Недоліки словесно-покрокового та графічного способів у вигляді схем опису алгоритмів програм

Спосіб опису	Недоліки
Словесно-покроковий	Незрозуміло, що є головним, а що другорядним (щось можна зрозуміти лише після індукції основного задуму). Важко відстежуються передачі керування Неможливість ефективного тестування.
Графічний у вигляді схем алгоритмів програм	Незрозуміло, що є головним, а що другорядним (щось можна зрозуміти лише після індукції основного задуму). Важко записувати коментарі. Для розуміння схеми алгоритмів необхідно доповнювати досить довгими текстовими описами, які мають

містити велику кількість тестових даних різних маршрутів обчислень.
Неможливість ефективного тестування.

До кінця 70-х років проектна процедура отримання алгоритмів базувалася на операційному (маршрутному) мисленні, яке закладається ще у школі математичними та фізичними навчальними дисциплінами. Операційний підхід не вимагає вільного володіння дедуктивним мисленням і ґрунтується на більш простому і вже освоєному індуктивному мисленні «від приватного до спільного». При операційному мисленні спочатку записуються послідовні дії головним основним маршрутом. Потім ці дії доповнюються операціями розгалуження (if), операціями безумовного переходу (go to) та додатковими діями інших маршрутів.
В результаті використання операційного підходу виходили програми (підпрограми) зі структурними елементами у вигляді операторів мови програмування та єдиною надскладною структурою "вся програма" типу "спагетті" - безліччю невпорядкованих передач управлінь вперед і назад (див. рис. 5.1). Все це можна зарахувати і до більшості текстових інструкцій.
Програми, алгоритми яких виходять операційним підходом, були практично несупроводжуваними. Навіть під час використання графічних схем алгоритмів їх потрібно було доповнювати досить довгими текстовими описами. Ці текстові описи мали містити величезну кількість окремих тестових даних для відстеження всіх маршрутів обчислень з метою розуміння алгоритму кожного окремого маршруту. Складність таких описів призводила до невідповідності коментарів обчислювальним процесам, а також невиправдано тривалого аналізу алгоритму з прорахунком траси рахунку по всіх маршрутах в процесі налагодження. При використанні схем алгоритмів програм поза документацією залишається перебіг думок проектувальника щодо отримання алгоритмів. Саме креслення схем алгоритмів програм займає досить багато часу. Величезне, астрономічне число обчислювальних маршрутів вимагало підготовки астрономічної кількості тестів. Програми були ненадійними, порівняно з нинішніми програмами. Налагодження програми у вигляді однієї нероздільної структури «спагетті» і довжиною всього 600 рядків займало час до півроку. Написання та налагодження такої програми за сучасних технологій проводиться програмістом за два-три робочі дні.
Відповідно до сучасних технологій програмування, описи алгоритмів у словесно-покроковій та графічній формах у вигляді схем алгоритмів програм практично не використовуються. Їх замінили самодокументовані тексти, що складаються із стандартних структур кодування.
Складання опису проектної процедури передували праці Дейкстри (з концепцією програмування без go to), однією з перших робіт з структурного кодування програм [9] і досвід програмування і викладання авторів.

5.3. ЗАГАЛЬНИЙ ОПИС ПРОЕКТНОЇ ПРОЦЕДУРИ

Принципи опису послідовності дій для алгоритмів та єврроритмів практично не відрізняються. В описах алгоритмів зазвичай є лише більша формалізація. При написанні програм програміст зазвичай використовує комп'ютер. Як правило, алгоритми більшості процедур програм є найпростішими. Зазвичай код таких процедур досвідчені програмісти пишуть одразу сидячи за монітором. Кодування навіть алгоритмічно складних процедур можна здійснювати з використанням комп'ютера, адже сучасний комп'ютер відмінно замінює папір та олівець! Просто треба паралельно у двох вікнах монітора розробляти документальний опис процедури та кодувати текст коду самої процедури. Навіть якщо код програми є само документованим, окремий документ міститиме детальний опис структури даних усієї програми, а також наочні тести, що детально характеризують весь алгоритм. Наявність такої документації спростить супровід програми.
Виконання проектної процедури починається з першого кроку, який полягає у повному з'ясуванні завдання зовнішньому рівні. Цьому допомагають набори тестових прикладів та модель «чорної скриньки». Набори тестових прикладів сприяють обліку всіх можливих випадків роботи алгоритму чи евроритму. Модель «чорного ящика» сприяє виявленню цільового призначення алгоритму, евроритму (інструкції) або іншої штучної системи, що розробляється. Модель «чорної скриньки» також допомагає виявляти вхідну та вихідну інформацію програм та інструкцій або визначити матеріальні, енергетичні та інформаційні входи та виходи інших штучних систем, що розробляються. Існують алгоритми та єврроритми, складання яких можна починати або з аналізу «чорної скриньки», або з підготовки

первинних тестів. Також відомі алгоритми, розробки яких до якісного завершення робіт доводилося багаторазово перемикатися як у роботу з «чорним ящиком», і на складання первинних текстів.
При розробці алгоритмів програм вхідна, проміжна та вихідна інформація характеризується структурою даних, розміщених в оперативній та зовнішній пам'яті, та інформацією, що відображається на екрані монітора.
При розробці євроритмів інструкцій вхідна, проміжна та вихідна інформація характеризується набором предметів (фізичних та документів) та їх станом, наприклад: порожній чайник; чайник, заповнений наполовину обсяг холодною водою; включений вимикач «мережа» у правому верхньому куті панелі; документ за формою № 5 із заповненою першою графою. Тут доречно доповнювати опис предметів їх малюнками. Враховуйте, що предмети змінюються у часі та просторі.
Первинні тестові приклади повинні включати як звичайні, і стресові набори тестових вхідних даних. Кожен стресовий набір тестових даних призначений виявлення реакції у випадках, наприклад неправильних дій користувача, поділу на нуль, виходу значення допустимі кордону, невідповідності інструкції стану справ тощо. буд. Будь-який набір тестових даних має містити опис результату.
Другий крок складання євроритму або алгоритму починається з розробки узагальнюючих тестів або узагальнюючого тесту, що включають як найменший набір тестів всі випадки з первинних тестів.
На основі узагальнюючих тестів, а також виявлених входів та виходів системи готується наочний узагальнюючий тест або кілька наочних узагальнюючих тестів. Наочний тест повинен відображати ланцюжок перетворення інформації від вихідної інформації через проміжну інформацію до результуючої інформації. Найголовніша вимога до узагальнюючого тесту – його наочність у поданні порядку зміни інформації. Відомо, що з вивченні геометрії успіх розв'язання завдання визначається наочністю малюнків. Наочні геометричні малюнки зазвичай виходять багатоваріантним їх побудовою, і потрібно придбання деяких навичок у розвиток мистецтва їх виконання. Усе це стосується і складання наочних узагальнюючих тестів. Наочність узагальнюючих тестів досягається правильним вибором способу відображення інформації, що забезпечує сприйняття порядку перетворення інформації.
Наочні уявлення зазвичай грунтуються моделі типу абстракції даних, викладеної в гол. 1. Однією її форм може бути функціональна модель як набору діаграм потоків даних (ДПД). Інша форма може відповідати малюнку з раціональним способом розміщення його площі масивів з їх іменами, значеннями елементів, значеннями індексів елементів, значеннями і іменами простих змінних, з'єднаними стрілками у напрямку передачі інформації. Ще однією формою є форма, близька до траси рахунку, що показує зміни значень змінних на етапах процесу. Можливі інші форми, наприклад таблиці правил.
Наголосимо на важливості роботи з даними. Дані багато в чому визначають алгоритми. При вирішенні однієї і тієї ж задачі, вибір різних структур даних призводить до алгоритмів або євроритмів, що абсолютно розрізняються.
Подальше виконання проектної процедури полягає у отриманні на підставі моделі абстракції даних моделі процедурної абстракції, також викладеної в гол. 1.
Будь-які алгоритми або євроритми повинні складатися тільки зі стандартних структур, кожна з яких має суворо один інформаційний вхід і один інформаційний вихід. Використання інших (нестандартних) структур призводить або до подовження опису, або до неможливості тестування (через неможливо великого обсягу необхідних тестів), або до втрати зрозумілості.
Втрата зрозумілості відбувається через те, що в неструктурованому алгоритмі чи євроритмі одні й ті самі частини алгоритму чи євроритму за одних даних виконують одне, а за інших — інше. Тому частини неструктурованих алгоритмів чи евроритмів неможливо однозначно характеризувати засобами природної мови.
Структурі СЛІД в інструкціях і програмах відповідає суворо одна дія.
Далі проектна процедура виконується ітеративними кроками: до досягнення елементарних дій (елементарних операторів мови програмування або елементарних операцій) окремі структури СЛІД, з яких складається опис будь-якого алгоритму або евроритму, декомпозуються з дотриманням принципу від загального до одного з трьох стандартних структур (рис. 5.2).): ланцюжок Слідкувань; Ланцюжок АЛЬТЕРНАТИВ; ПОВТОРЕННЯ.
У разі довгого алгоритму вичерпання інформації узагальнюючого тесту готуються нові узагальнюючі тести під нові завдання структури.

*Мал. 5.2.*Виявлення виду чергової структури під час виконання проектної процедури розробки функціональних описів

З трьох виявлених структур будь-яка структура містить у собі одну або кілька нових структур виду СЛІД з більш приватними діями. Ці нові СЛІДКИ можуть піддаватися декомпозиції на наступній ітерації виконання проектної процедури.

Отже, описи послідовності процесів для алгоритмів і евроритмов мало розрізняються, тому розробку розглянемо разом. Процес кодування програм, суміщений з документуванням, передбачає різноманітні роботи і потребує особливих знань, тому кодування програм розглянемо окремо. Зазначимо, що алгоритм ряду програм (наприклад, математично не складних) та код програм народжуються одночасно, причому народжуються на основі думок програміста. Звідси випливає, що ознаки алгоритмічних типових структур та порядок їх деталізації, описані в даному підрозділі, є одними й тими самими для складання алгоритмів, евроритмів та програм.

На кожній ітерації проектної процедури доводиться вирішувати завдання: «А яка саме із трьох структур буде виявлена?» При вирішенні цього завдання необхідну інформацію можна отримати лише з аналізу узагальнюючих тестів. Аналіз тестів щодо пошуку найголовнішої на даний момент структури є для початківців дуже непростою справою. «Розкріпляти мислення» допомагає набір ознак структур, викладений у табл. 5.2 а також набір евристичних прийомів, викладений далі.

Таблиця 5.2.

Зведена таблиця характеристик структур та ознак структур

Структура	Характеристики	Ознака
СЛЕДЖЕННЯ	Описується або простими поширеними реченнями природної мови, або реченнями без присудка (наприклад, «Навантаження меблів», «Рішення квадратного рівняння»)	Відповідає строго одній дії
Ланцюжок слідів	Являє собою ланцюжок із послідовно виконуваних дій	Послідовно виконувані різнорідні дії
Ланцюжок альтернатив:		Одна або кілька дій, кожна з яких виконується за певної умови або не виконується взагалі
проста АЛЬТЕРНАТИВА	Описується конструкцією: «Якщо виконується якась умова, то виконується СЛІД 1»	

АЛЬТЕРНАТИВА із двох дій	Описується конструкцією: «Якщо виконується якась умова, то виконується СЛІД 1, в іншому випадку виконується СЛІД 2»
ВИБІР	Є ланцюжком з більш ніж двох найпростіших альтернатив з однією дією
ПОВТОРЕННЯ:	Багаторазово виконувана дія (але обов'язково кінцеве число разів). Повторенням відповідають думки: «Ця дія має бути виконана п'ять разів»; «Ця дія виконується багаторазово до настання такої події». Ознаками ПОВТОРЕНЬ також є змінна кількість АЛЬТЕРНАТИВ, будь-яка думка про повернення «назад», щоб повторити якісь дії. Часто головний загальний процес виду повторення прихований у контексті «і т. д.» або «і т. п.», «це зовсім просто», або навіть у крапках «...»
ПОВТОРЕННЯ «ДО»	Описується конструкцією: «До виконання якоїсь умови багаторазово виконувати СЛІД»
ПОВТОРІННЯ «ПОКИ»	Описується конструкцією: «Поки виконується якась умова, багаторазово виконувати СЛІД»
НЕУНІВЕРСАЛЬНЕ ПОВТОРЕННЯ	Забезпечує задану кількість повторень

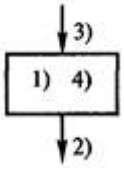
Набір евристичних прийомів:

1. «Хороші наочні ілюстрації – запорука успіху!».
2. "Думай від спільного до приватного!".
3. "Загальний процес визначає роботу приватних!".
4. «Це не головний процес, ви загрузли в частковості!».
5. «Не забувай запроваджувати нові терміни (імена змінних)!».
6. «Виділивши головну дію, ви вже вирішуєте простішу задачу!».
7. «Якщо закінчилася інформація в узагальнюючих тестах, то готуйте нові узагальнюючі тести для вирішення нових приватних завдань!».
8. «Якщо в процесі декомпозиції потрібно описати процес виходу з якоїсь точки опису в якусь іншу, це означає, що були неправильно виконані попередні деталізації через неправильне виявлення найбільш загальної дії і потрібно коректно переробити попередню роботу!».
9. «Іноді чергова деталізація не виходить через неусвідомлену потребу щодо введення допоміжної змінної (прапора події), що характеризує, чи відбулася раніше якась подія!».

Кожне СЛІДЖЕННЯ відповідає строго одній дії. Головне у діях – дієслово. Основне, що необхідно виконати при описі окремої структури СЛІД: в описі повинна міститися лише одна думка. Будь-яка структура виду СТЕЖЕННЯ може бути описана простою поширеною пропозицією природної (російської) мови. Наприклад, "Наступна дія полягає в завантаженні меблів в автомобіль". Однак враховуючи те, що у разі складання алгоритмів програм суть типових структур пояснюється самими операторами мови програмування, застосовується більш короткий опис у вигляді неповної речення без присудка. У разі підлягає утворюють від дієслів. Наприклад, "Навантаження меблів", "Рішення квадратного рівняння", "Введення даних".

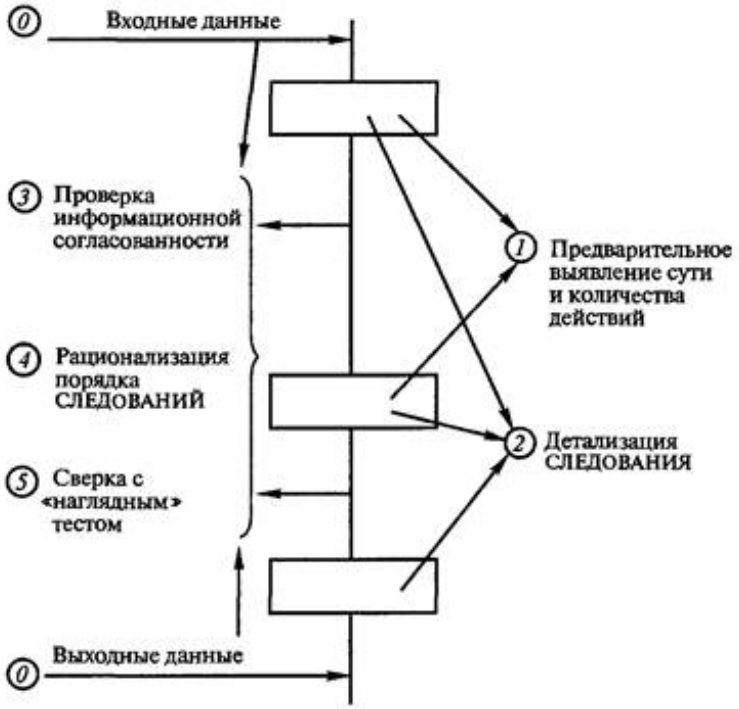
Порядок деталізації одиночного СЛІДЖЕННЯ з використанням моделі «чорної скриньки» показано на рис. 5.2; 5.3:

1) попереднє виявлення суті дії;
2) виявлення вихідний інформації, бо вихід визначає вхід, а чи не навпаки;
3) виявлення вхідної інформації;



*Мал. 5.3.*Модель «чорної скриньки»

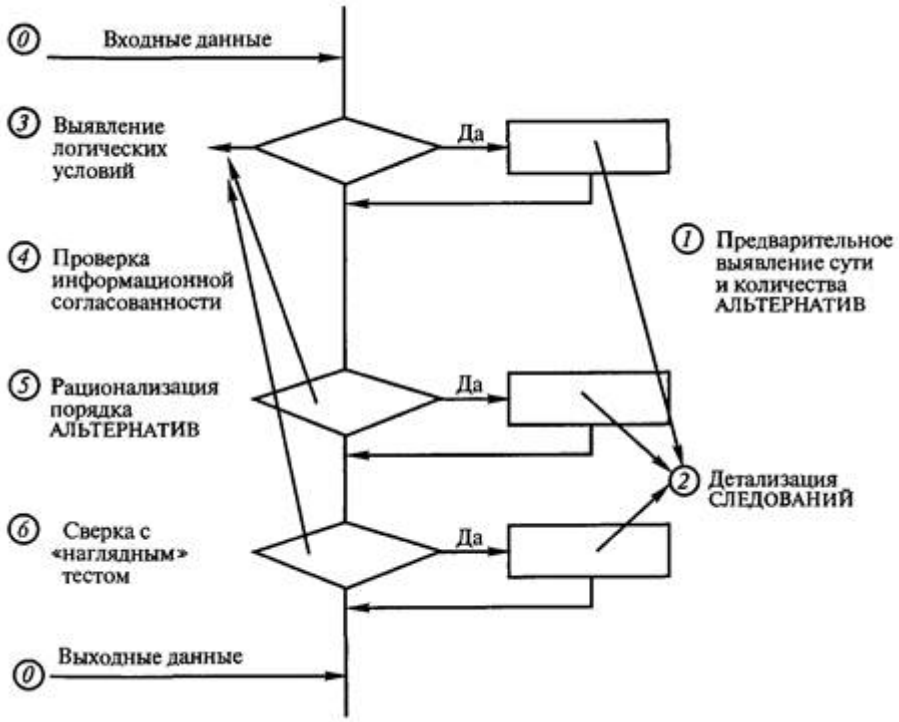
4) запис уточненого коментаря суті дії чи одного заключного елементарного оператора.
При описі фізичних та технічних систем необхідно використовувати повнішу модель «чорної скриньки» (див. рис. 2.2).
Ланцюжок слідів відповідає однозначному опису послідовності дій.
Ознакою ланцюжка Слідування є ніколи не порушується послідовність дій (послідовність Слідування).
Ланцюжок слідів відповідає послідовність простих пропозицій і складносурядні пропозиції, які краще перетворити на прості пропозиції.
Порядок деталізації ланцюжка слідів показано на рис. 5.4:
1) попереднє виявлення суті дій кожного з СЛІД, визначення кількості СЛІД;
2) деталізація кожного зі СЛІДІВ як одиночного СЛІДЖЕННЯ;



*Мал. 5.4.*Порядок дій при деталізації структури Ланцюжок слідів

3) перевірка інформаційної узгодженості всіх СЛІД, а також вхідної та вихідної інформації всього ланцюжка СЛІД;
4) раціоналізація порядку СЛІД (зосередження СЛІД, що співпрацюють в інформаційному обміні);
5) звіряння з узагальнюючими тестами.
Окремі Слідування структури Ланцюжок Слідкувань надалі можуть бути декомпозовані Ланцюжком Слідкувань (більш приватних). Однак зустрічаються окремі структури СЛІДЖЕННЯ, які не можуть бути декомпозовані ланцюжком СЛІДЖЕНЬ. Такі СЛЕДЖЕННЯ можуть бути описані або структурою виду ланцюжка альтернатив (розгалуження), або структурою виду повторення (ЦИКЛ).
Ознакою ланцюжка альтернатив або вибору є одна або кілька дій, кожна з яких виконується за певної умови або не виконується взагалі (обов'язково кінцеве число різних дій за різних умов).
Ланцюжок Альтернатив, в окремому випадку, може складатися з однієї або декількох простих альтернатив з однією дією. Пропозиція найпростіша АЛЬТЕРНАТИВА з однією дією має таку конструкцію: «Якщо виконується якась умова, то виконується такий процес (СТЕЖЕННЯ)». (Інакше СЛІДКА пропускається.) Пропозиція Альтернативу з двох дій має наступну конструкцію: «Якщо виконується якась умова, то виконується СЛІД 1, в іншому випадку виконується СЛІД 2». В принципі структура Альтернативу з двох дій еквівалентна ланцюжку з двох найпростіших альтернатив з однією дією. При деталізації процесів, що включають більше двох альтернатив, може бути отримана єдина структура-ВИБІР у вигляді ланцюжка послідовно записаних структур АЛЬТЕРНАТИВУ з однією дією.
Тут слід зазначити, що від зовні схожого ланцюжка слідів, кожне слідування якої є найпростішою

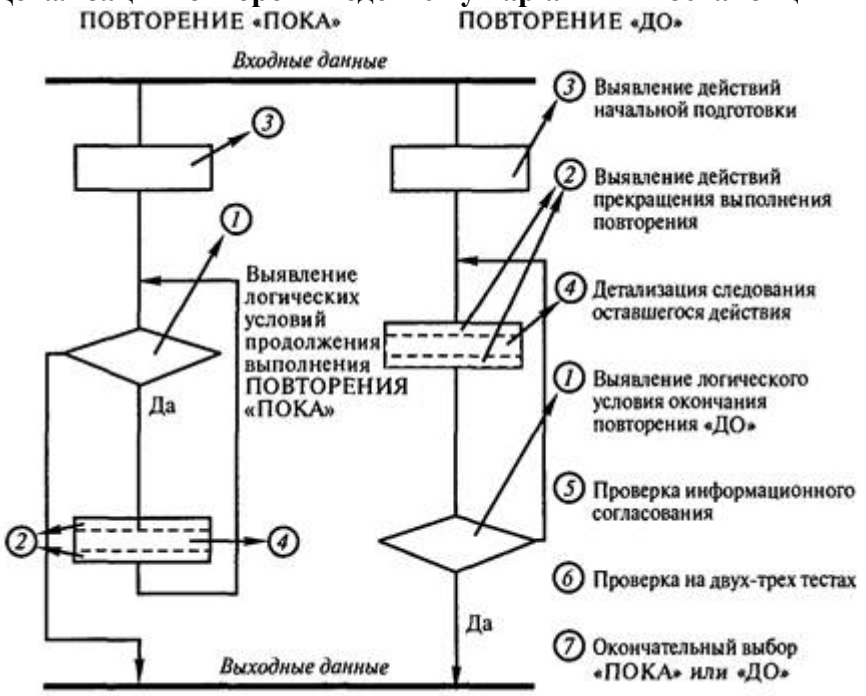
альтернативою з однією дією, структура ВИБІР відрізняється такою властивістю, що при виконанні всього ланцюжка альтернатив може виконуватися лише одне з альтернативних слідств або жодного з. *прикла*думов альтернатив у разі структури ВИБІР:
Якщо А < В, то виконати СЛІД 1;
Якщо А = В, то виконати СЛЕДЖЕННЯ2;
Якщо А > В, то виконати СЛІД.
Альтернатива з однією дією можна здійснити дострокове припинення процесу виконання алгоритму в тому випадку, який відповідає виявленню умов неможливості правильного подальшого виконання алгоритму або евроритму.
Деталізація всіх наступних структур передує нульову дію - запис на початку і в кінці вхідних і вихідних даних, виявлених в процесі деталізації попередніх їм СЛІД.
Порядок деталізації Ланцюжки Альтернатива показано на рис. 5.5:
1) попереднє виявлення суті дії кожного зі СЛІДІВ альтернативних дій, визначення кількості таких СЛІДЖЕНЬ;



*Мал. 5.5.*Порядок дій при деталізації ланцюжка АЛЬТЕРНАТИВ, згідно з проектною процедурою розробки функціональних описів

2) деталізація кожного зі СЛІДІВ як одиночного СЛІДЖЕННЯ;
3) виявлення та запис логічної умови виконання кожного з альтернативних СЛІДІВ;
4) перевірка інформаційної узгодженості всіх СЛЕДЖЕНЬ та логічних умов у ланцюжку, а також вхідної та вихідної інформації всього ланцюжка альтернатив;
5) раціоналізація порядку альтернатив;
6) перевірка виконання всіх маршрутів на тестах, отриманих із узагальнюючого тесту.
Ознакою ПОВТОРЕННЯ є багаторазово виконувана дія (але обов'язково кінцеве число разів). ПОВТОРЕННЯМ відповідають думки: «Ця дія має бути виконана п'ять разів»; «Ця дія виконується багаторазово до настання такої події». Ознаки ПОВТОРЕНЬ — змінна кількість АЛЬТЕРНАТИВ, будь-яка думка про повернення «назад», щоб повторити якісь дії. Часто головний більш загальний процес виду повторення ховається в контексті «і т. д.» або «і т. п.», «це зовсім просто», або навіть у крапках «...».
Пропозиція виду ПОВТОРЕННЯ може бути записана або у формі ПОВТОРЕННЯ «ДО» (ЦИКЛ «ДО»), або у формі ПОВТОРЕННЯ «ПОКИ» (ЦИКЛ «ПОКИ»).

Пропозиція ПОВТОРЕННЯ «ДО» має таку конструкцію: «До виконання якоїсь умови багаторазово виконувати СЛІД».
Пропозиція ПОВТОРЕННЯ «ПОКИ» має таку конструкцію: «Поки виконується якась умова, багаторазово виконувати СЛІД».
Різниця між пропозиціями повторення «ДО» та повторення «ПОКИ» полягає в тому, що, згідно з першою пропозицією, дія СТЕЖЕННЯ має бути виконана хоча б один раз, а згідно з другим, — СЛІДЖЕННЯ може не виконуватися жодного разу.
Структура НЕУНІВЕРСАЛЬНЕ ПОВТОРЕННЯ або просто забезпечує задану кількість повторень якогось процесу або виконання якогось процесу при значенні змінної циклу, значення якої змінюється за правилами арифметичної прогресії.
Порядок деталізації ПОВТОРЕНЬ показано на рис. 5.6.
Деталізація повторень ведеться у варіантній постановці



*Мал. 5.6.*Порядок дій при деталізації ПОВТОРЕНЬ

1) виявлення та запис логічної умови завершення ПОВТОРЕННЯ «ДО» або умови продовження виконання ПОВТОРЕННЯ «ПОКИ»;
2) виявлення процесів припинення виконання повторення;
3) виявлення дій СЛІДІ підготовки підготовки ПОВТОРЕННЯ;
4) деталізація СЛЕДЖЕННЯ дії, що залишилася, як одиночного СЛІДЖЕННЯ;
5) перевірка інформаційної узгодженості всіх СЛІД, логічних умов, а також вхідної та вихідної інформації всього ПОВТОРЕННЯ;
6) перевірка на 2-3 тестах, отриманих з узагальнюючого тесту;
7) остаточний вибір варіанта реалізації ПОВТОРЕННЯ у вигляді структури ПОВТОРЕННЯ «ПОКИ» або у вигляді структури ПОВТОРЕННЯ «ДО».

5.4. РЕКОМЕНДАЦІЇ ПОЧИНАЮЧИМ З СКЛАДАННЯ ОПИСІВ АЛГОРИТМІВ І ЕВРОРИТМІВ

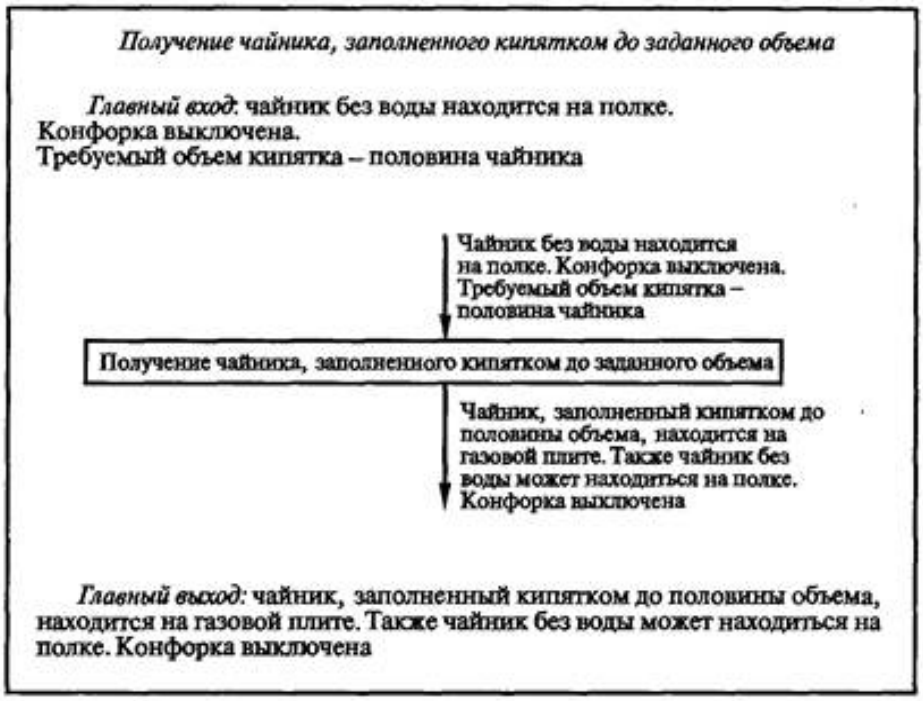
Перші спроби роботи з проектною процедурою вимагають величезної кількості аркушів паперу, але це необхідно, тому що дозволяє уважно розглянути окремий аркуш і безболісно переробити невдалі деталізації. Після досягнення деякого досвіду можна все більшу частину роботи виконувати без її запису на окремих аркушах. Проте перша робота виконується суворо окремих листах. Особливо це

важливо у разі наявності досвідченого фахівця, який по аркушах документа зможе виявити дефекти навичок у учня, а також видати серію індивідуальних підказок, що навчається.
Виконання проектної процедури починається з підготовки та розгляду тестових прикладів для деталізації всього описуваного процесу у вигляді одного СЛІДЖЕННЯ.
На окремому аркуші складається безліч тестових прикладів, що охоплюють усі випадки обчислень чи дій з інструкції.
Паралельно з підготовкою тестів здійснюємо аналіз моделі «чорної скриньки». Беремо ще один чистий аркуш паперу, внизу аркуша записуємо терміни вихідних об'єктів та їх стану. Щодо алгоритмів визначаємо структуру даних: вводимо позначення змінних, послідовностей, векторів, матриць та визначаємо порядок розміщення в них інформації. Стосовно фізичних систем визначаємо розташування та стани об'єктів. Наприклад, чайник без води знаходиться на полиці. Результат дій - чайник, заповнений окропом до половини обсягу, знаходиться на плиті. Вихід визначає вхід, а чи не навпаки! Тому спочатку записуються результати дій і потім стану об'єктів до дій.
Далі у верхній частині листа записуються терміни вхідних об'єктів та стану вхідних об'єктів. У середині листа записується одна проста поширена пропозиція, що характеризує процес перетворення вхідної інформації у вихідну інформацію. Багаторазовим аналізом входу, виходу та суті пропозиції уточнюється вся інформація листа.
На основі сукупності тестів на окремому аркуші складаємо один або кілька узагальнюючих тестів. При цьому потрібно спроектувати раціональну форму ілюстрування: наприклад, малюнки проміжних станів даних процесу чи таблиці станів даних або фізичних об'єктів. Узагальнюючий тест необхідний розуміння суті функціонування процесу. На узагальнюючому тесті наводяться позначення всіх вхідних, вихідних та внутрішніх змінних чи станів фізичних об'єктів. З тесту мають бути видно всі процеси перетворення вхідної інформації на результат.
Навчальним, які перебувають у початковій стадії вивчення, рекомендується попередньо описати перебіг процесу за принципом: «Як можеш!» Якість опису має відповідати спробі пояснення процесу якійсь пересічній людині чи навіть дитині. Цей текст допоможе (з використанням ознак) відрізнити спільні дії від приватних.
Нарешті, приступаємо до складання семантичного скелета структурованого опису функціонування.
Під час навчання деталізації черговий структури слід здійснювати окремому аркуші. При деталізації ланцюжків слідів рекомендується орієнтувати лист вузькою стороною вгору, а при деталізації ланцюжків альтернатив і повторень лист краще орієнтувати широкою стороною вгору (це дозволить розміщувати на аркуші траси прорахунку тестів). Попередньо у верхній частині аркуша записується речення, яке було отримано раніше з деталізації кожного з цих процесів у вигляді одного СЛІДЖЕННЯ. Під ним записується вхідна інформація СЛІД, а внизу аркуша - вихідна інформація. Далі здійснюється сама деталізація, результати якої після перевірки на тестових прикладах та літературної обробки переносяться до чистовика опису алгоритму в цілому.
При деталізації Ланцюжки слідств рівномірно по вільній частині листа записуються пропозиції суті послідовно виконуваних дій (конкретний смисловий коментар до процесу). Далі здійснюється робота над кожним із них як над окремим СЛІДЖЕННЯМ. Далі здійснюється перевірка інформаційної узгодженості слідування та раціоналізується їх порядок, уточнюється суть СЛІД. При перевірці необхідно переконатися, що для наступних СЛІДІВ дані вже були визначені попередніми СЛІДЖЕННЯМИ.
При деталізації ланцюжка альтернатив рівномірно по вільній частині листа записуються (у кількості альтернативних дій) конструкції у вигляді потрібної кількості наступних послідовних речень: слово «Якщо», кілька чистих рядків для поля умови, слова «то виконується дія», далі залишається кілька чистих рядків для пропозиції СЛІД (конкретний смисловий коментар процесу).
Після деталізація всіх записаних СЛЕДЖЕНЬ як одного СЛІДЖЕННЯ здійснюється запис умов виконання альтернативних процесів. Далі здійснюється перевірка інформаційної узгодженості входу та виходу кожного зі СЛІД, їх умов виконання, а також вхідної та вихідної інформації всього ланцюжка альтернатив. Ланцюжок лише з двох альтернатив може бути описаний пропозицією виду: «Якщо виконується умова (конкретна умова виконання дії по ТО), то виконується процес (конкретний змістовий коментар процесу), інакше виконується інший процес (конкретний змістовий коментар процесу).

При деталізації ПОВТОРЕНЬ на вільній частині листа записуються слова обох заготовок для ПОВТОРЕННЯ «ДО» та ПОВТОРЕННЯ «ПОКИ». Кожна заготівля починається з чистих рядків для майбутнього СЛІДУ визначення підготовки повторюваних дій. Далі для ПОВТОРЕННЯ «ДО»: записується рядок «До виконання умови закінчення процесу, що повторюється»; залишається кілька чистих рядків для запису умови закінчення дії, що повторюється; записується рядок «багаторазово виконується така дія:...». Для ПОВТОРЕННЯ «ПОКИ» записується рядок «Поки виконується умова»; залишається кілька чистих рядків для запису умови продовження виконання дії, що повторюється; записується рядок «багаторазово виконується така дія:...». Під цими рядками на обох заготовках залишаються чисті рядки для СЛІД, що визначає закінчення повторення процесу і СЛІД, що багаторазово виконується дії. Заповнення заготовок здійснюється в наступному порядку: спочатку умова закінчення (продовження для повторення «ПОКИ») виконання дії, що повторюється; потім записується СЛІД, що визначає закінчення повторення процесу; Потім — СЛІД визначення визначення повторюваних дій і в останню чергу — СЛІД багаторазово виконуваної дії. Далі здійснюється перевірка інформаційної узгодженості входу та виходу кожного зі СЛІД, умов виконання дії, а також вхідної та вихідної інформації всієї структури. Якщо на початок деталізації був ясно, який із варіантів повторень раціональніше деталізувати насамперед, то послідовно деталізуються обидві заготовки. Остаточний варіант відбирається шляхом їх порівняння за критеріями стислості та зрозумілості.
Після закінчення деталізації структурних окремих частин документа необхідно зробити його збирання. При складанні зазвичай потрібне літературне редагування документа як єдиного цілого. Після смислової та літературної обробки результат досліджень процесу переноситься до чистовика. Головний принцип — передай іншим хід своїх думок так, щоб вони змогли зрозуміти, але при цьому не слід перестаратися в деталях. Зайві коментарі можуть викликати роздратування читача та посилити нерозуміння.

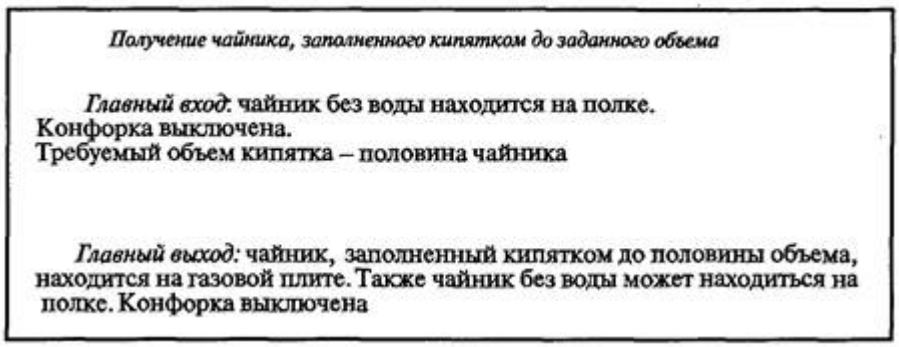
5.5. ПРИКЛАД РОЗРОБКИ ОПИСУ ПРОЦЕСУ «КИП'ЯЧЕННЯ ВОДИ У ЧАЙНИКУ»

Нижче показано покрокове виконання проектної процедури на прикладі розробки опису процесу кипіння води в чайнику. Доповніть цей опис наочними малюнками на аркуші 1 самостійно.
Аркуш 2.Аналіз процесу як одного СЛІДЖЕННЯ.
Первинний опис суті дії: "Кип'ятіння води в чайнику".
*Вихід:*Чайник, наповнений окропом до половини обсягу, знаходиться на газовій плиті. Конфорку вимкнено.
*Вхід:*Чайник без води знаходиться на полиці. Конфорку вимкнено. Необхідний об'єм окропу – половина чайника.
Остаточний опис суті дії: "Отримання чайника, заповненого окропом до заданого обсягу".
Працюємо із тестами. З тестів з'ясовуємо, що одержати окріп неможливо без води, сірників та газу. Сірники можуть закінчитися у процесі запалення газу. Може закінчитись вода в процесі заповнення чайника. Також може закінчитись газ у магістралі. Приймаємо, що виявлення даних фактів буде здійснено у процесі виконання інструкції, тому інформацію про наявність сірників, води та газу виключаємо зі складу вхідної інформації.
Нами отримано СЛІД (рис. 5.7).
Аркуш 3- Містить зображення процесів інструкції в наочній формі. Розробте його самостійно.
Аркуш 4- може містити опис процесу «Отримання чайника, заповненого окропом до заданого об'єму», виконаний у будь-який доступний спосіб.



*Мал. 5.*7. Деталізація із застосуванням графічного зображення «чорної скриньки»

Аркуш 5.Декомпозиція процесу "Отримання чайника, заповненого окропом до заданого обсягу".
Спочатку на лист переносимо інформацію попередньої структури СЛІД, одержуємо макет листа, представлений на рис. 5.8.
Далі, виходячи з міркувань, що для цього процесу необхідно виконати ряд послідовних дій, отримуємо макет аркуша з ланцюжком стежень, представлений на рис. 5.9.
Після деталізації кожного з прямування, перевірки інформаційної узгодженості та уточнення суті прямування в ланцюжку отримуємо макет аркуша, зображений на рис. 5.10.
Перевірка інформаційного узгодження виявила, що дія 2 може бути раніше дії 1 з міркувань економії газу. Чайник не можна ставити на конфорку до запалювання газу через небезпеку закопчення денця кіптявою сірника. Аналогічно перевірялася послідовність інших процесів.
Дія 1 може бути декомпозірована ще однією ланцюжком СЛІД, зображеної на рис. 5.11.



*Мал. 5.8.*Початковий вид аркуша 5

Получение чайника, заполненного кипятком до заданного объема

Главный вход: чайник без воды находится на полке.
Конфорка выключена.
Требуемый объем кипятка – половина чайника

Действие 1. Заполнение чайника водой

Действие 2. Зажигание газовой горелки

Действие 3. Перемещение чайника с полки на конфорку

Действие 4. Кипячение воды в чайнике

Действие 5. Выключение конфорки

Главный выход: чайник, заполненный кипятком до половины объема, находится на газовой плите. Также чайник без воды может находиться на полке. Конфорка выключена

*Мал. 5.9.*Вид аркуша 5 після попереднього виявлення суті дій

Дія 1 «Заповнення чайника водою» може бути декомпозіроване ще однією ланцюжком слідок, зображеної на рис. 5.11.
Дія 4 "Наповнення чайника водою", показане на рис. 5.11 декомпозується ПОВТОРЕННЯМ, представленим на рис. 5.12.

Лист 5

Получение чайника, заполненного кипятком до заданного объема

Главный вход: чайник без воды находится на полке.
Конфорка выключена.
Требуемый объем кипятка – половина чайника

Вход: Чайник без воды находится на полке
Действие 1. Заполнение чайника водой
Выход: Чайник с надетой крышкой, заполненный до половины объема холодной водой, находится на полке

Вход: Конфорка выключена
Действие 2. Зажигание газовой горелки
Выход: Конфорка включена

Вход: Чайник, заполненный до половины объема холодной водой, находится на полке. Конфорка включена
Действие 3. Перемещение чайника с полки на конфорку
Выход: Чайник, заполненный до половины объема холодной водой, находится на конфорке. Конфорка включена

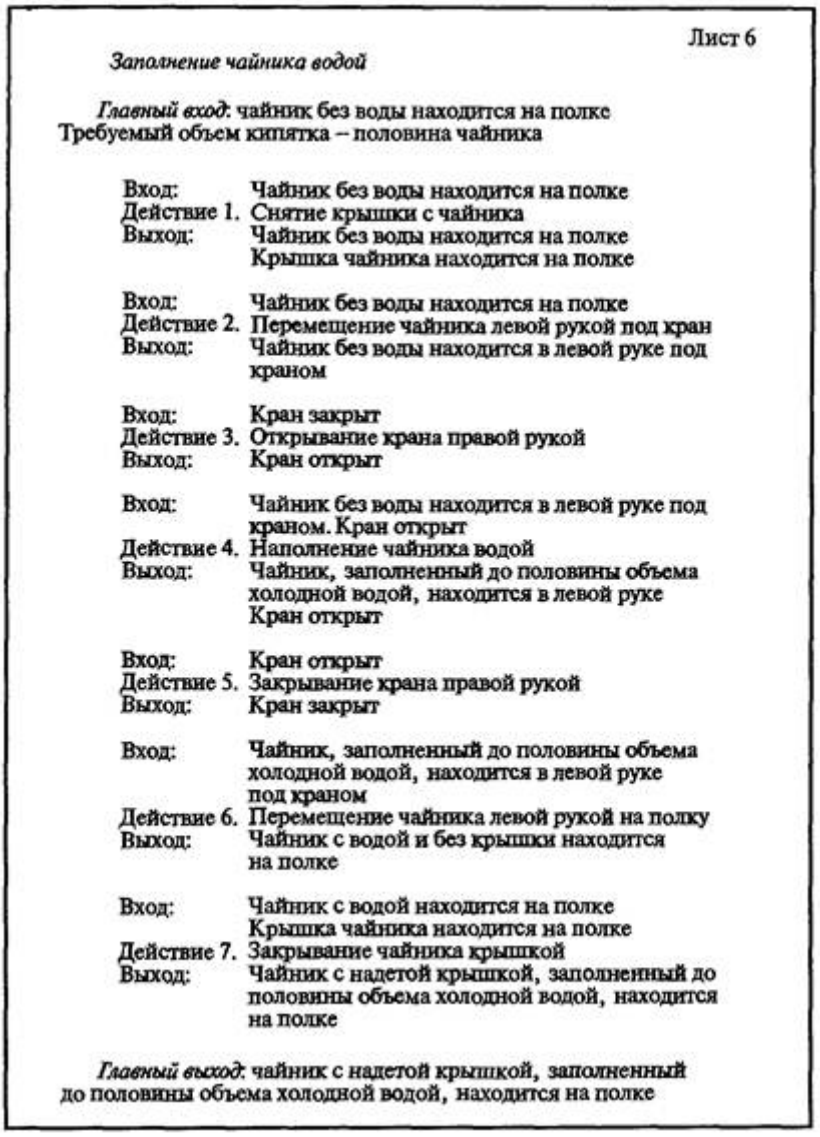
Вход: Чайник, заполненный до половины объема холодной водой, находится на конфорке Конфорка включена
Действие 4. Кипячение воды в чайнике
Выход: Чайник, заполненный до половины объема горячей водой, находится на конфорке Конфорка включена

Вход: Конфорка включена
Действие 5. Выключение конфорки
Выход: Конфорка выключена

Главный выход: чайник, заполненный кипятком до половины объема, находится на газовой плите. Также чайник без воды может находиться на полке. Конфорка выключена

*Мал. 5.10.*Остаточний вигляд листа

Декомпозиція ланцюжком СЛІД дії «Аналіз процесу заповнення чайника на форс-мажорні обставини» представлена на рис. 5.13.



Лист 6

Заполнение чайника водой

Главный вход: чайник без воды находится на полке
Требуемый объем кипятка – половина чайника

Вход: Чайник без воды находится на полке
Действие 1. Снятие крышки с чайника
Выход: Чайник без воды находится на полке
Крышка чайника находится на полке

Вход: Чайник без воды находится на полке
Действие 2. Перемещение чайника левой рукой под кран
Выход: Чайник без воды находится в левой руке под краном

Вход: Кран закрыт
Действие 3. Открывание крана правой рукой
Выход: Кран открыт

Вход: Чайник без воды находится в левой руке под краном. Кран открыт
Действие 4. Наполнение чайника водой
Выход: Чайник, заполненный до половины объема холодной водой, находится в левой руке
Кран открыт

Вход: Кран открыт
Действие 5. Закрывание крана правой рукой
Выход: Кран закрыт

Вход: Чайник, заполненный до половины объема холодной водой, находится в левой руке под краном
Действие 6. Перемещение чайника левой рукой на полку
Выход: Чайник с водой и без крышки находится на полке

Вход: Чайник с водой находится на полке
Крышка чайника находится на полке
Действие 7. Закрывание чайника крышкой
Выход: Чайник с надетой крышкой, заполненный до половины объема холодной водой, находится на полке

Главный выход: чайник с надетой крышкой, заполненный до половины объема холодной водой, находится на полке

*Мал. 5.11.*Деталізація СЛІД «Заповнення чайника водою», виявленого на рис. 5.10

Дія «Форс-мажорне завершення інструкції» представляє ланцюжок СЛІД, показаному на рис. 5.14. Зазначимо, що вихідна інформація аварійного завершення інструкції визначається вхідною інформацією всієї інструкції (див. рис. 5.7).

Лист 7

Наполнение чайника водой

Главный вход: чайник без воды находится в левой руке под краном. Кран открыт

Действие. Повторение до заполнения половины объема чайника холодной водой; многократно выполнять действие «Анализ процесса заполнения чайника на форс-мажорные обстоятельства».

Анализ действия «Анализ процесса заполнения чайника на форс-мажорные обстоятельства» как одиночного СЛЕДОВАНИЯ

Вход: Вид струи воды
Действие. Анализ процесса заполнения чайника на форс-мажорные обстоятельства
Выход: Нет

Главный выход: чайник, заполненный до половины объема холодной водой, находится в левой руке. Кран открыт

*Мал. 5.12.*Декомпозія повторенням дії 4 «Наповнення чайника водою» (див. рис. 5.11)

Лист 8

Анализ заполнения чайника на форс-мажорные обстоятельства

Главный вход: вид струи воды

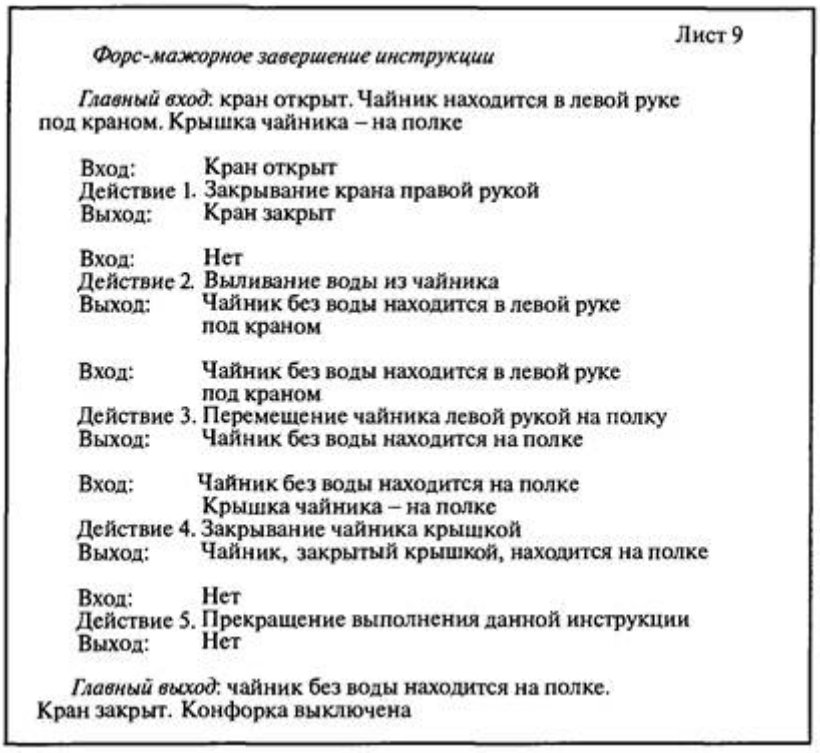
Действие «Анализ процесса заполнения чайника на форс-мажорные обстоятельства» представляет собой ЦЕПОЧКУ АЛЬТЕРНАТИВ из одной альтернативы:
ЕСЛИ течет из крана грязная вода или недостаточная струя (нет струи),
ТО
форс-мажорное завершение инструкции

Анализ действия «Форс-мажорное завершение инструкции», как одиночного СЛЕДОВАНИЯ

Вход: Кран открыт. Чайник находится в левой руке под краном
Действие. Форс-мажорное завершение инструкции
Выход: Чайник без воды с надетой крышкой находится на полке

Главный выход: чайник без воды находится на полке. Конфорка выключена

*Мал. 5.13.*Декомпозиція ланцюжкового слідства дії «Аналізу процесу заповнення чайника на форс-мажорні обставини» (див. рис. 5.12)



Лист 9

Форс-мажорное завершение инструкции

Главный вход: кран открыт. Чайник находится в левой руке под краном. Крышка чайника – на полке

Вход: Кран открыт
Действие 1. Закрывание крана правой рукой
Выход: Кран закрыт

Вход: Нет
Действие 2. Выливание воды из чайника
Выход: Чайник без воды находится в левой руке под краном

Вход: Чайник без воды находится в левой руке под краном
Действие 3. Перемещение чайника левой рукой на полку
Выход: Чайник без воды находится на полке

Вход: Чайник без воды находится на полке
Крышка чайника – на полке
Действие 4. Закрывание чайника крышкой
Выход: Чайник, закрытый крышкой, находится на полке

Вход: Нет
Действие 5. Прекращение выполнения данной инструкции
Выход: Нет

Главный выход: чайник без воды находится на полке. Кран закрыт. Конфорка выключена

*Мал. 5.14.*Декомпозиція дії «Форс-мажорне завершення інструкції» представляє ланцюжок СЛІД.

Дія 2 (див. рис. 5.10) «запалювання газового пальника» декомпозується циклом: доки не запалилася конфорка або не виявлено форс-мажорні обставини, багаторазово виконувати дію «Запалювання конфорки». Подальший розвиток алгоритму дозволив виявити структури, викладені далі.
Дія (СТЕЖЕННЯ) «Запалювання конфорки» є ланцюжком СЛІД: «Запалювання сірника», «Включення газу», «Підпалювання газу», «Відключення газу при невдачі», «гасіння сірника», «викидання сірника». Форс-мажорними обставинами є відсутність газу або закінчення сірників. Їм відповідає логічна змінна. Дія (СТЕЖЕННЯ) «Відключення газу при невдачі» є альтернативою: якщо газ не запалився, а сірник прогорів, то необхідно закрити газ.
Тут виявлено попередню недоробку з вихідними даними дії 2. Адже після закінчення дії 2 конфорка може бути як включена, так і не включена. Виправити ситуацію можна двома способами. За першим способом за дією 2 можна вставити найпростішу альтернативу з однією дією: якщо виявлено форс-мажорні обставини, аварійно завершити інструкцію. Відповідно до другого способу, дії 3, 4, 5 можуть бути представлені одним СЛІДОМ, всередині якого повинна знаходитися Ланцюжок Альтернатив послідовного виконання кожного з колишніх дій 3, 4, 5 за умови відсутності виявлення форс-мажорних обставин.

5.6. ПРИКЛАД ОПИСУ ПРОГРАМИ «РЕДАКТОР ТЕКСТІВ»

Нижче наведено приклад опису програми «Редактор текстів», складений одним із учнів. У прикладі наводиться спочатку зовнішня функціональна специфікація, потім внутрішня специфікація.
Програма «Редактор текстів» призначена для створення нових та коригування існуючих текстових файлів MS DOS у діалоговому (користувач-ЕОМ) режимі роботи. ЕОМ формує екран із вікном, у якому відображено ділянку тексту з текстового файлу (макет екрану відповідає внутрішньому редактору програми Norton Commander). Користувачеві забезпечується можливість вставки в текст у вікні екрана будь-якого символу клавіатури за символом, позначеним на екрані курсором. Виняток становить ряд символів, які є ознаками команд керування або незадіяними символами (наведено список символів). Після подачі користувачем команди запису, всі зміни тексту, здійснені користувачем, записуються у файл.

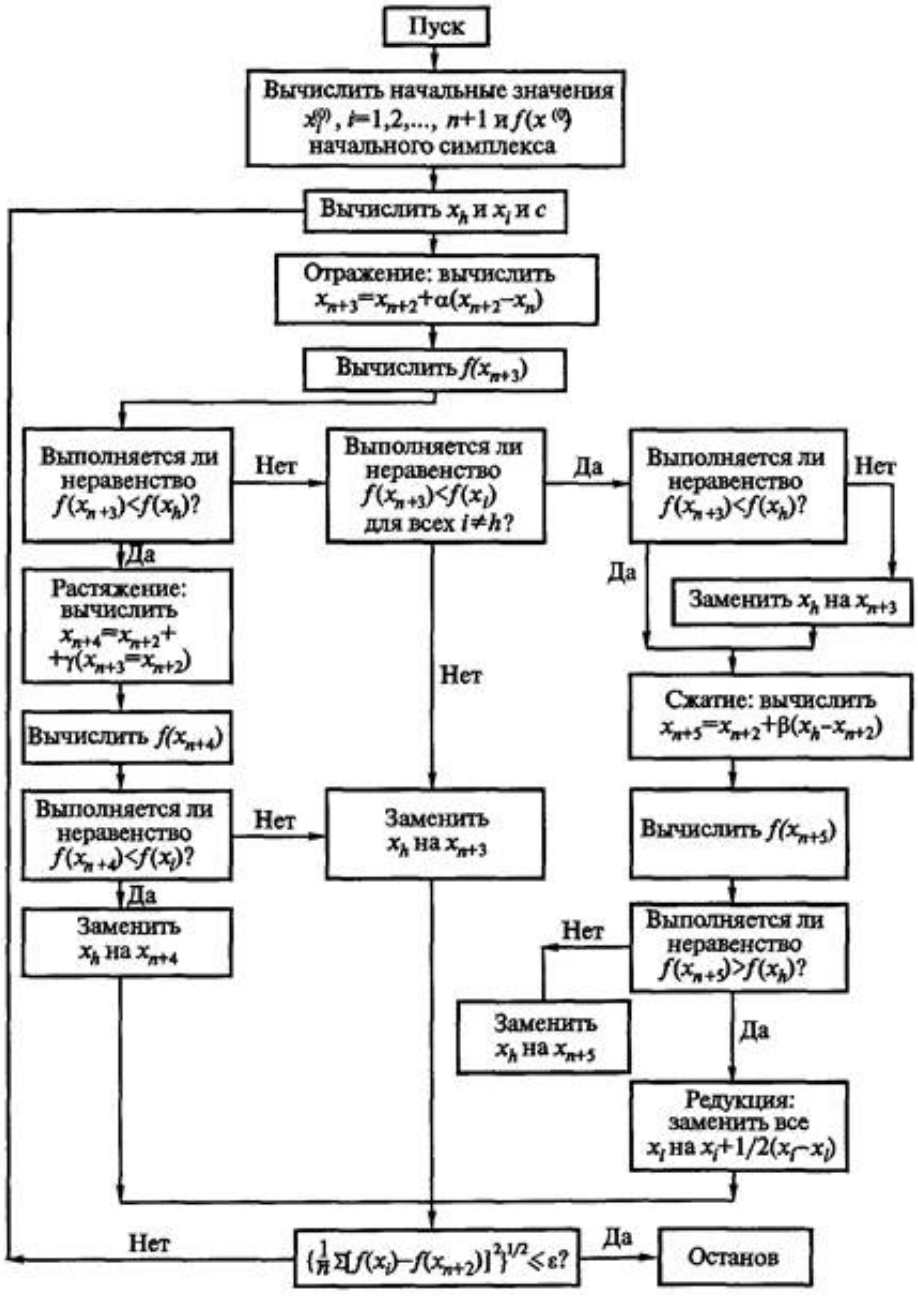
Основний принцип роботи редактора текстів полягає у перенесенні рядків тексту з необхідних ділянок файлу спочатку в буферний масив пам'яті довжиною 65535 байт (символів) з подальшим копіюванням необхідних рядків з буферного масиву у вікно екрану.
Запуск програми здійснюється командою із зазначенням імені файлу, що редагується. Далі, доки не буде вказано коректне ім'я файлу, може почати багаторазово виконуватися алгоритм «Запит користувача на введення або коригування імені файлу».
Потім задаються початкові значення структурованої змінної "Система координат", в якій є поля: "Положення курсору щодо файлу"; «Положення курсору щодо буферного вікна редактора»; "Положення буферного вікна редактора щодо файлу".
Після цього здійснюється очищення буферного масиву редактора рядкових змінних з 5 * 23 = 115 рядків довжиною по 225 символів.
Далі при параметрі "Перший рядок файлу" виконується алгоритм "Завантаження рядків файлу, починаючи із зазначеного рядка в буферний масив редактора". Потім до подачі користувачем однієї з команд завершення редагування із збереженням інформації (або збереження) виконується головний цикл програми. Нарешті, якщо було дано команда завершення зі збереженням інформації, то інформація з буферного масиву переписується файл. Виконання програми завершується очищенням екрана.
Контроль імені файлу, що редагується, полягає в наступному. Якщо файл із вказаним ім'ям відсутній на диску, виводиться попереджувальне повідомлення про створення нового "порожнього" файлу. Якщо користувач не вказав ім'я файлу, що редагується або відмовився працювати зі створеним «порожнім» файлом, то відбувається аварійне завершення програми з поясненням причини завершення.
Усередині головного циклу програми виконується ряд із трьох послідовних дій. "Алгоритм відображення" відображає на екрані 23 рядки тексту з буферного масиву, починаючи із заданого рядка. Далі встановлюється курсор дисплея на позицію екрана. Здійснюється введення коду натиснутої кнопки. Якщо код натиснутої клавіші відповідає керуючій клавіші, то виконується одна з альтернативних дій щодо виконання команди, яка відповідає цій клавіші. В іншому випадку вставляється символ у текст.

5.7. РЕФАКТОРИНГ АЛГОРИТМІВ І ЕВРОРИТМІВ

Алгоритм Нелдера-Міда є широко відомим і застосовується як алгоритм прямого пошуку локального екстремуму речових функцій від 2 до 6 речових змінних.
Наступний абзац містить фрагмент тексту з книги Д. Хіммельблау [26], в якому міститься частина опису алгоритму Нелдера - Міда (методу багатогранника, що деформується).
У методі Нелдера і Міда мінімізується функція n незалежних змінних з використанням n + 1 вершин багатогранника, що деформується, в En. Кожна вершина може бути ідентифікована x вектором. Вершина (точка) в En, в якій значення f(x) максимально, проектується через центр ваги (центроїд) вершин, що залишилися. Покращені (нижчі) значення цільової функції знаходяться послідовною заміною точки з максимальним значенням f(x) більш «хороші» точки, доки знайдено мінімум f(x). Початковий багатогранник зазвичай вибирається у вигляді регулярного симплексу (але це не обов'язково) з точкою на початку координат. Процедура відшукання вершини в En, у якій f(x) має найкраще значення, складається з наступних операцій:
Відображення — проектування x(k)h через центр тяжіння відповідно до співвідношення x(k)n+3 = x(k)n+2 + α(x(k)n+2 — x(k)h), де α > 0 є коефіцієнтом відбиття; x(k)n+2 — центр тяжіння, x(k)h — вершина, у якій функція f(x) приймає найбільше з n + 1 її значень на k етапі.
Розтягування виконується, якщо f(x(k)n+3) ≤ f(x(k)i), то вектор (x(k)n+3 - x(k)n+2) розтягується відповідно до співвідношення x(k)n+4 = x(k)n+2 + γ(x(k)n+3 - x(k)n+2), де γ - коефіцієнт розтягування. Якщо f(x(k)n+4) < f(x(k)i), то x(k)h замінюється на x(k)n+4 і процедура продовжується з кроку 1 (k = k + 1), інакше x(k)n замінюється на x(k)n+3 процедура триває з кроку 1 (k = k + 1).
Стиснення — якщо f(x(k)n+3) > f(x(k)i) для всіх i ≠ h, то вектор (x(k)h — x(k)n+2) стискається відповідно до формули x(k)n+5 = x(k)n+2 + β(x(k)h — x(k)n+2), де 0 < β < 1 являє собою коефіцієнт стиснення. Потім x(k)h замінюється на x(k)n+5 і перетворюється на крок 1 (k = k + 1).
Редукція — якщо f(x(k)n+3) > f(x(k)h), всі вектори (x(k)i — x(k)1), i = 1...n + 1 зменшуються 2 рази з відліком від x(k)1 відповідно до формули x(k)i = x(k)i + 0,5(x(k)i — x(k)1), i = 1...n + 1. Потім повертаємось до кроку 1 для продовження пошуку на k + 1 кроці.

Критерій закінчення пошуку, використаний Нелдером і Мідом, був перевіркою умови середнього квадратичного відхилення функцій f(x(k)i) від довільного малого числа ε.

Хіммельблау скористався традиційним математичним стилем викладу опису алгоритму. Алгоритм має структуру виду "спагетті", що видно зі схеми алгоритму, розробленої автором (рис. 5.15). Схема як «спагетті» — це приховані go to, які ми виявляємо у тексті програми, розробленої автором. Він же наводить графічні малюнки принципу розрахунку точок та графічний малюнок послідовності просування кращих точок симплексу по кроках методу при вирішенні тестового завдання. Витративши 11 сторінок тексту, навівши текст неструктурованої програми на 12 сторінках, автор книги [26] не зумів зрозуміло описати свій алгоритм.



*Мал. 5.15.*Схема алгоритму, розроблена Д. Хіммельблау

Порівняйте спосіб подання опису алгоритму, складений автором [22], та функціональний опис алгоритму після рефакторингу. Функціональний опис алгоритму наведено нижче.

Особливістю алгоритму є просування до екстремуму цілою хмарою пробних точок, яка умовно названа симплексом (деформованим багатогранником). Загальна кількість точок у симплексі дорівнює збільшеному на одиницю числу змінних пошуку. Просування до екстремуму не по лінії від однієї точки

до іншої, а всередині якогось безлічі пробних точок забезпечило нереагування на дрібні дефекти цільової функції та проходження щодо «широких» ярів.
Інформація, що вводиться:
n - число змінних функції, що мінімізується;
X0 = (x01, x02, ..., x0n) - точка першого початкового наближення;
h - крок регуляризації симплексу;
εx - похибка знаходження екстремуму за параметрами.
Робота алгоритму починається з початкової підготовки симплексу точок. Після виконання процедур регуляризація симплексу та розрахунку значень цільової функції у всіх точках симплексу має вигляд

$$\begin{array}{c} 1 \\ 2 \\ 3 \\ \dots \\ n+1 \end{array}\begin{pmatrix} x_1^0 & x_2^0 & x_3^0 & \dots & x_n^0 \\ x_1^0+h & x_2^0 & x_3^0 & \dots & x_n^0 \\ x_1^0 & x_2^0+h & x_3^0 & \dots & x_n^0 \\ \dots & \dots & \dots & \dots & \dots \\ x_1^0 & x_2^0 & x_3^0 & \dots & x_n^0+h \end{pmatrix}\begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \\ \dots \\ Q_{n+1} \end{pmatrix}$$

Процедура регуляризації симплексу полягає в копіюванні у всі поля значень аргументів симплексу значення вектора X0, і далі проводяться зміни значень діагональних компонентів векторів з номерами від 2 до n + 1 шляхом їх збільшення на крок регуляризації симплексу h.

У симплексі Q1, Q2, Q3, ..., Qn+1 — це розраховані значення функції, що мінімізується, при відповідних за рядком значеннях аргументів мінімізованої функції.
Далі до виконання умови закінчення пошуку здійснюються ітерації (кроки) пошуку. Умовою закінчення пошуку цього є невіддаленість від кращої точки симплекса інших точок симплекса більш як на e2x.
Кожна ітерація починається з знаходження номерів l і k відповідно кращої і гіршої за значенням функції точок симплекса. Далі здійснюється розрахунок точки Xц - положення центру мас усіх точок симплексу за винятком найгіршої точки 1. Це пізнє поліпшення алгоритму. Д. Хіммельблау не виключав найгіршої точки.

$$x_{iц} = \frac{i}{n}\sum_{j=1}^{n+1}(x_{ji}),$$

де n - кількість точок у симплексі; i – номер компоненти вектора X(i = 1, 2, ..., n); j – номер точки в симплексі.
Вважаються нульовими значення xkj доданків сум, де k — номер найгіршої точки.
Працюючи методу кожної з ітерації може обчислюватися одне з спеціальних пробних точок: a — точка відбиття, p — точка розтягування, y — точка стискування. Крапки обчислюються за формулами:
Xα = Xц + α(Xц - Xk), Xβ = Xц + β(Xц - Xk), Xγ = Xц + γ(Xц - Xk).
Доцільно ці схожі формули реалізувати однією функцією.
Значення коефіцієнтів α = 1, β = 0,5 та γ = 2 підібрані експериментальним шляхом (в оригіналі опису алгоритму ці значення можна виявити лише з тексту програм). Розрахунок точок α, β, γ доцільно здійснювати однією процедурою з параметром у вигляді значень коефіцієнтів α, β, γ.
Після розрахунку точки Xц обчислюється пробна точка. Далі виконується одна з альтернативних дій.
Якщо Qα ≤ Q1, виконуються дії досягнення сильного успіху. Якщо Q1 ≤ Qα < Qk, є слабкий успіх, а точка α записується на місце k-точки. Якщо Qα ≥ Qk, виконуються дії відсутності успіху.
Розраховується Xβ та Qβ. Далі, якщо Qβ ≤ Qk, то точка β записується на місце k-точки, інакше, якщо точка β гірша за точку k, виконується процедура редукції симплексу та процедура розрахунку значення функції в точках симплексу.
Дії досягнення сильного успіху. Розраховується Xγ і Qγ. Найкраща точок α або β записується на місце найгіршої k-точки симплексу.
Події відсутності успіху. Розраховується Xβ та Qβ. Далі виконується дія зі зміни симплексу за відсутності успіху.
Дія зі зміни симплексу за відсутності успіху є альтернативою: якщо Qβ ≤ Qk, то точка β записується на місце k-точки, інакше, якщо точка β гірша за точку k, виконується процедура редукції симплексу.

При виконанні процедури редукції симплексу всі точки симплексу стягуються до кращої точки симплексу на половину свого попереднього видалення і далі виконується процедура розрахунку значень цільової функції у всіх точках симплексу.

5.8. Кодування типових структур на мовах програмування

Зазвичай розробку алгоритмів програм поєднують із кодуванням тексту програми. Окреме від програмування написання алгоритмів практично не відрізняється від написання інструкцій.
Кодування програми має здійснюватись лише з використанням стандартних структур! Заборонено використання міток, операторів безумовного переходу на мітку (go to), операторів дострокового виходу із структури break!
При кодуванні мовою С оператор break може використовуватися лише при кодуванні структури switch.
При використанні іншої процедурно-орієнтованої мови програмування (не Pascal) необхідно попередньо закодувати використовуваною мовою програмування всі описані в цьому підрозділі стандартні структури без зміни їх логіки!
Так, при програмуванні мовою С структура УНІВЕРСАЛЬНИЙ ЦИКЛ — «ДО» включатиме операцію «!» (НЕ):

```
/* підготовка циклу */
do
{
/* Тіло циклу */
...
}
while (! (L));
```

У наведеній вище структурі ненульове значення змінної L відповідає закінченню виконання циклу, а чи не його продовженню виконання, як у оператора мови програмування! Використання «лінією» операції (!) не подовжить програму. Сучасні компілятори автоматично інвертують логічну умову завершення циклу.
Структурі СЛІД В програмах можуть відповідати: виконання всієї програми; виклик процедури.
Згідно зі стандартом проекту, АЛЬТЕРНАТИВА має чотири конструкції. Розглянемо їх запис мовою програмування Pascal.

Конструкція для однієї альтернативи:

```
if L then begin
{Дія при L=True}
...
end;
```

Конструкція для двох альтернатив:

```
if L then begin
{Дія при L=True}
...
end
else
begin
{Дія при L=False}
...
End;
```

Перший варіант конструкції для кількох альтернатив (ВИБОРУ):

```
if L1 then Begin
{Дія при L1=True}
end;
...
if L2 then
begin
{Дія при L2=True}
...
end;
```

```
if L3 then
begin
{Дія при L3=True}
...
end;
```

Другий варіант конструкції для кількох альтернатив (ВИБОРУ):

```
Switch: = 0;
L1: = ...;
L2: = ...;
L3: = ...;
...
if L1 then Switch := 1;
if L2 then Switch := 2;
if L3 then Switch := 3;
...
case Switch of
1:begin
{Дія при L1=True}
...
end;
2:begin
{Дія при L2=True}
...
end;
3:begin
{Дія при L3=True}
...
end;
else
begin
{Виведення повідомлення про помилкове кодування модуля}
...
end;
end; {End of Case}
```

Розглянемо запис варіантів кодування структури АЛЬТЕРНАТИВУ мовою програмування З.

Конструкція для однієї альтернативи:

```
if (L)
{
/*Дія при L ≠ 0*/
...
}
```

Конструкція для двох альтернатив:

```
if (L)
{
/*Дія при L ≠ 0*/
...
}
else
{
/*Дія при L = 0*/
...
}
```

Перший варіант конструкції для кількох альтернатив (ВИБОРУ)

```
if (L1)
{
/*Дія при L1 ≠ 0*/
...
}
else if (L2)
```

```
{
/*Дія при L2 ≠ 0*/
...
}
else if(L3)
{
/*Дія при L3 ≠ 0*/
...
}
...
}
```

Другий варіант конструкції для кількох альтернатив (ВИБОРУ):

```
Selector = 0;
L1 = ...;
L2 = ...;
L3 = ...;
...
if (L1) Selector = 1;
else if (L2) Selector = 2;
else if (L3) Selector = 3;
...
switch (Selector)
case 1:
/*Дія при L1≠ 0*/
...
break;
case 2:
/*Дія при L2≠ 0*/
...
break;
case 3:
/*Дія при L3 ≠ 0*/
...
break;
default:
/*Виведення повідомлення про помилкове кодування модуля*/
exit (-1);
} /*Кінець switch*/
```

Права конструкція відповідає дуже складній логіці умов. У найпростіших випадках допускається спрощене кодування (перший приклад Pascal, другий Q:

```
if a > b then x:=y+3 else x:=y+6; {Мова Pascal}
if (a > b) x=y+3; else x = y +6; /*Мова C*/
```

ВИБІР із двох і більше АЛЬТЕРНАТИВ не можна кодувати за допомогою вкладення інших структур найпростіших АЛЬТЕРНАТИВ через велику ймовірність помилок.

Порядок деталізації структур АЛЬТЕРНАТИВУ:

1) залежно кількості альтернативних дій записуються всі оператори структури;
2) визначаються самі альтернативні дії як СЛІДКИ;
3) записуються логічні умови альтернативних действий;
4) перевіряється інформаційна узгодженість логічних умов та дій;
5) на кількох текстових прикладах здійснюється перевірка. ПОВТОРЕННЯ в програмуванні називаються циклами.

Зазвичай стандартом проекту передбачено низку конструкцій циклів. Неуніверсальний ЦИКЛ-ДО має дві конструкції та використовується для завдання заданого числа повторень. Розглянемо їх запис мовою програмування Pascal.

Конструкція по зростанню:

```
for i:=3 to 5 do begin
{тіло циклу i=3,4,5}
...
end;
```

Конструкція зі спадання:
```
for i:=5 downto 3 do begin
{тіло циклу i=5,4,3}
...
end;
```

Розглянемо запис варіантів кодування структури неуніверсального ЦИКЛ-ДО мовою програмування З.

Конструкція по зростанню:
```
for (i=3; i<=5; i++)
{
/ * Тіло циклу i = 3,4,5 * /
...
}
```

Конструкція зі спадання:
```
for (i=5; i>=3; i--)
{
/ * Тіло циклу i = 5,4,3 * /
...
}
```

Тут і – змінна циклу. Зазвичай, ці цикли не вимагають після кодування додаткового тестування. Універсальні цикли мають конструкції ЦИКЛ-ДО та ЦИКЛ-ПОКИ. Їх запис на мові Pascal наведено нижче:

Універсальний ЦИКЛ-ПОКИ:
```
{Підготовка}
while L do
begin
{Тіло циклу}
...
end;
```

Універсальний цикл ЦИКЛ-ДО:
```
{Підготовка}
repeat
{Тіло циклу}
...
until L;
```

Нижче наведено запис тих самих структур мовою С:

Універсальний ЦИКЛ-ПОКИ:
```
/*Підготовка*/
while (L)
{
/*Тіло циклу*/
...
}
```

Універсальний ЦИКЛ-ДО:
```
/*Підготовка*/
do
{
/* Тіло циклу */
...
}
while (! (L))
```

Тут L логічний вираз. Його значення True є умовою продовження виконання ЦИКЛ-ПОКИ або умовою закінчення виконання ЦИКЛ-ДО. Підготовка та тіло циклу є СЛЕДЖЕННЯМИ. Тіло циклу виконується стільки разів, як і весь цикл. Ознакою ЦИКЛ-ПОКИ є можливість не виконання тіла циклу жодного разу. При рівноцінності з двох конструкцій ЦИКЛ-ДО і ЦИКЛ-ПОКИ вибирають ту, запис якої коротше.
Взагалі ЦИКЛ-ДО можна закодувати структурою ЦИКЛ-ПОКИ, якщо у підготовці записати деякі дії із тіла циклу. З ЦИКЛУ-ДО виходить ЦИКЛ-ПОКИ за його охоплення структурою АЛЬТЕРНАТИВУ.
Порядок декомпозиції циклів:
1) набирається "порожній" текст оператора циклу;
2) записується логічне умова продовження ЦИКЛ-ПОКИ чи завершення ЦИКЛ-ДО (у своїй виявляється змінна циклу);
3) декомпозується та дія тіла циклу, яка змінює логічну умову до невиконання умови ЦИКЛ-ПОКИ або до виконання умови ЦИКЛ-ДО;
4) деталізується СЛІД «Підготовка циклу»;
5) деталізується решта дії тіла циклу як СЛІД;
6) проводиться перевірка інформаційної узгодженості всіх елементів циклу;
7) проводиться перевірка правильності роботи циклу за допомогою безмашинного розрахунку траси виконання тестів із трикратним (або більше), одноразовим виконанням циклу та взагалі без виконання.
У текстах програм може використовуватися ще одна обчислювальна структура – РЕКУРСІЯ. Ознакою цієї структури є наявність рекурсивних формул та обчислень. Все, що робить рекурсія, можна продати за допомогою циклів і масивів. Однак якщо мова програмування допускає рекурсію, її використання може скоротити код програми. Рекурсія завжди дуже ретельно коментується.

5.9. МЕТОДИКА РОЗРОБКИ АЛГОРИТМІВ ПРОГРАМ

Розглянемо порядок роботи з методики розробки структурованих алгоритмів на прикладі. Нехай потрібно розробити програму розв'язання квадратного рівняння, яке має вигляд
ax2 + bx + c = 0.
Робота за методикою починається з повного з'ясування задачі. Цьому допомагає застосування моделі «чорного ящика», розробка форм програми, що вводиться і виводиться (наприклад, у вигляді макетів екрану), підготовка первинних тестових прикладів. Немає для всього різноманіття завдань точний порядок виконання цих дій. Дані дії часто доводиться виконувати паралельно, перемикаючись з дії на дію з вичерпанням можливостей розвитку поточної дії та відкриття можливостей розвитку чергової дії.
Спочатку алгоритм повинен представляти одну типову структуру СЛІД (одна дія зі змістом виконати всі дії програми, наприклад, програма нарахування зароблені плати, але не програма нарахування зароблені плати та/або рішення квадратного рівняння).
Дивлячись на тести та зображення моделі «чорної скриньки» (див. рис. 5.3), деталізуємо весь алгоритм як одну СЛІД (послідовно виконувану дію) у порядку: а) попередній запис сенсу дії «чорної скриньки»; б) вихідна та/або виведена інформація; в) вхідна та/або інформація, що вводиться; г) визначається дія в «чорній скриньці» (одна пропозиція).
При розробці алгоритмів програм вхідна, проміжна та вихідна інформації характеризуються структурою даних. Важливим є порядок розміщення значень у масивах, імена та значення констант опису розмірностей масиву, імена та значення змінних, що характеризують поточні значення використовуваного розміру масиву, ім'я та порядок зміни змінної індексу поточного елемента масиву. Форма введеної та виведеної на екран або друк інформації може бути показана макетами екранів або документів.
Первинні тестові приклади повинні включати як звичайні, і стресові набори тестових вхідних даних. Кожен стресовий набір тестових даних призначений виявлення реакції у випадках. Наприклад: невірних дій користувача, поділу на нуль, виходу значення за допустимі межі тощо. буд. Будь-який набір тестових даних має містити опис результату.
Досліджуючи «чорну скриньку» стосовно розв'язання квадратного рівняння, можемо записати попередній коментар суті всіх дій програми: «Програма розв'язання квадратного рівняння виду a*x*x + b*x + c = 0».

Далі з'ясовується, що ще не виявлено вихідну інформацію «чорної скриньки», тому необхідно перейти до підготовки тестів, що допоможе продовжити роботу з «чорною скринькою». У даному випадку підготувати тести допоможе аналіз завдання.
Отже, нехай відома "шкільна" формула розв'язання квадратного рівняння виду ax2 + bx + c = 0.
Відомо також, що спочатку треба обчислити дискримінант рівняння D:
D = b2 - 4ac.
Навіть якщо забули про випадок негативності дискримінанта нічого страшного немає. Записуємо формулу рішення:

$x1 = (-b - \sqrt{D})/(2a);$

$x2 = (-b + \sqrt{D})/(2a).$ Нам відомо, якщо D < 0, то з негативного числа не можна витягувати квадратний корінь. Тому згадуємо, що за негативного дискримінанта немає коренів. Ще виявляємо факт особливого випадку, якому відповідає факт при D = 0 наявності двох рівних коренів. Ще відомо, що ділити на нуль не можна, а за a = 0 маємо саме цей випадок. У цьому випадку вихідне квадратне рівняння перетворюється на лінійне рівняння:
bx+c=0.
Рішення рівняння, що вийшло, буде наступним:
x = (-c) / b.
Це рішення можливе лише у разі a = 0 і (одночасно) b ≠ 0. У разі a = 0 і (одночасно) b = 0 і (одночасно) c ≠ 0 лінійне рівняння не має рішення.
Аналізуючи вихідне рівняння, з'ясовуємо, що у випадку a = 0 і (одночасно) b = 0 і (одночасно) c = 0 рівняння має безліч рішень (коріння x1 і x2 — будь-які числа).
Складемо наочну таблицю правил розв'язання квадратного рівняння (табл. 5.3).
Таблиця 5.3
Наочна таблиця правил розв'язання квадратного рівняння

№ п/п	a	b	з	d	Варіант розв'язання
1	a ≠ 0	Будь-яке	Будь-яке	d > 0	Два різні корені
2	a ≠ 0	Будь-яке	Будь-яке	d = 0	Два рівні корені
3	a ≠ 0	Будь-яке	Будь-яке	d < 0	Немає рішення
4	a = 0	b ≠ 0	Будь-яке	Ні	Є корінь лінійного рівняння
5	a = 0	b = 0	c ≠ 0	Ні	Немає рішення
6	a = 0	b = 0	з = 0	Ні	Безліч рішень

У табл. 5.3 немає поєднань значень, які ще виявлено. Тепер можна визначити вихідну інформацію «чорної скриньки», яка видається у п'яти варіантах:
1) рівняння має безліч рішень (коріння x1 і x2 — будь-які числа);
2) значення двох різних коренів x1 та x2;
3) значення двох рівних коренів у вигляді x1 і доповнює написи про два рівні корені;
4) напис не має рішення;
5) значення одного кореня x1 із написом, що рівняння є лінійним.
Тип змінних, у яких розміщуються вихідні значення коренів x1 і x2, - речовий (Real). Тепер визначаємо вхідну інформацію. З вихідного рівняння випливає, що вхідною інформацією є значення трьох коефіцієнтів a, b, c речовий типу (Real). У ході аналізу формул було встановлено, що значення трьох коефіцієнтів a, b, c можуть набувати будь-яких значень, що було не очевидно до аналізу формул розв'язання рівняння (наприклад, випадок a = 0).
Імена змінних будуть досить мнемонічними, якщо дотримуватися прийнятих у математиці позначень.
Остаточний коментар суті дій усієї програми: «Програма розв'язання квадратного рівняння a*x*x + b*x + c = 0 з довільними значеннями коефіцієнтів a, b, c типу речовий». Факт довільності значень коефіцієнтів a, b, c на етапі попереднього виявлення суті дії «чорної скриньки» ще виявлено.
Зрештою, готуємо тестові приклади.
Сукупність тестів для всіх виявлених випадків розв'язання квадратного рівняння:
1) при a = 0, b = 0, c = 0 безліч рішень (коріння x1 і x2 — будь-які числа);
2) при a = 2, b = 3, c = -2 значення двох різних коренів x1 = -2 і x2 = 0,5;

3) при a = 1, b = 4, c = 4 значення двох рівних коренів у вигляді x1 = x2 = -2 і виведення доповнюючого напису про два рівні корені;
4) при a = 2, b = 5, c = 4 виведення напису «немає рішення»;
5) при a = 0, b = 2, c = -8 значення одного кореня x1 = 4 з написом, що рівняння є лінійним;
6) при a = 0, b = 0, c = 2 виведення напису "немає рішення". Тепер можна відразу написати фрагмент програми, що відповідає виконаній роботі:

```
Program Kvadrat;
{ Програма розв'язання квадратного рівняння
виду a * x * x + b * x + c = 0 з довільними
значеннями коефіцієнтів a, b, c типу
речовий }
Uses
Crt, Dos;
Var
a, b, c: Real; {Коефіцієнти квадратного рівняння}
xl, x2: Real; {Корні квадратного рівняння}
begin
end.
```

Шляхом компіляції фрагмента програми можна перевірити коректність синтаксису. Тепер підготуємо макет зображення на екрані монітора (рис. 5.16). На макеті зображення екрана монітора символами □ позначені поля введення інформації, а символами ■ поля виведення інформації.
Зазвичай, що виводиться на екран інформація не містить імен змінних, але в даному випадку прийняті в математиці імена доцільно відобразити на екрані.
Макет зображення монітора також є тестом. На основі первинних тестових прикладів та організації вхідної та вихідної інформації готується або один тест, або кілька узагальнюючих тестів. Усі складені тести на вирішення квадратного рівняння увійшли в узагальнюючий тест.
Розробка наочних тестів. Для найпростішого алгоритму розв'язання квадратного рівняння первинні тести разом з математичними формулами вже мають наочність. Продовжимо складання програми розв'язання квадратного рівняння.
Спочатку потрібно виконати дії, визначені макетом зображення на екрані монітора, так як для виконання алгоритму розв'язання квадратного рівняння необхідні вихідні дані як коефіцієнтів a, b, c. Однак введення має передувати виведення інформації, що пояснює призначення програми і суть чергової порції даних, що вводиться. Це визначає первинну деталізацію одиночне СЛІДЖЕННЯ в ланцюжок СЛІД. Первинне одиночне СТЕЖЕННЯ не може містити вхідний та вихідний інформації. Вхідна інформація повинна бути введена, або присвоєна. Вихідна інформація після завершення програми нікого не цікавить. Отже, нижче представлено ланцюжок послідовних дій.

```
Программа решения квадратного уравнения
вида a*x*x + b*x + c = 0 с произвольными значениями
коэффициентов a, b, c типа вещественный

Укажите значение коэффициента a = □□□□□□□□□□□□□□□
Укажите значение коэффициента b = □□□□□□□□□□□□□□□
Укажите значение коэффициента c = □□□□□□□□□□□□□□□

Решается квадратное уравнение
■■■■■■■■■*x*x + ■■■■■■■■■■■■*x + ■■■■■■■■■■ = 0:

     Варианты строк вывода решения уравнения:
два различных корня x1 = ■■■■■■■■■■  x2 = ■■■■■■■■■■
два равных корня x = ■■■■■■■■■■
уравнение не имеет решения
уравнение линейное x = ■■■■■■■■■■
бесчисленное множество решений уравнения (корни - любые числа).
```

*Мал. 5.16.*Макет зображення екрану монітора

```
ClrScr; {Очищення екрана}
{Виведення інформації про призначення програми}
WriteLn ('Програма розв'язання квадратного рівняння');
WriteLn ("виду a * x * x + b * x + c = 0 з довільні", "ми значеннями");
```

```
WriteLn ('коефіцієнтів a, b, c типу речовий');
WriteLn;
{Введення значень коефіцієнтів a, b, c}
Write ('Вкажіть значення коефіцієнта a =');
ReadLn (a); {Введення a}
Write ('Вкажіть значення коефіцієнта b=');
ReadLn (b); {Введення b}
Write ('Вкажіть значення коефіцієнта з =');
ReadLn (c); {Введення з}
{ Виведення перевірочно-протокольної інформації про введені значення коефіцієнтів a, b,
c}
WriteLn;
WriteLn ('Вирішується квадратне рівняння');
WriteLn (a:10:4, '*x*x + ', b:10:4, '* x + ', з:10:4, '= 0:'); { Саме рішення квадратного
рівняння }
WriteLn;
Write ('Для завершення програми натисніть');
WriteLn ('будь-яку клавішу...');
Repeat until KeyPressed; { Цикл очікування натискання будь-якої клавіші }
```

Дія «Очищення екрану» — артефакт реалізації, а не завдання, але наскільки приємніше оглядати екран без зайвої інформації. Також артефактом реалізації з'явилися два останні оператори, які забезпечують зручність перегляду результатів роботи програми. За відсутності цих операторів та знаходження в оболонці Turbo Pascal для перегляду результатів роботи програми довелося б вручну перейти у вікно результатів.

Написання операторів WriteLn, Write, ReadLn не викликало труднощів завдяки заздалегідь підготовленому макету зображення екрана монітора.

Дія «Само рішення квадратного рівняння» представлена коментарем та порожнім рядком, що наголошує на факті додавання дій у майбутньому. Це пояснюється тим, що рішення та виведення результатів рішення багатоваріантні, а, отже, дію «Само рішення квадратного рівняння» не можна уявити ланцюжком Слідування. Багатоваріантність передбачає керуючі структури.

Уточнюємо коментарі, оператори виводу Write та WriteLn.

Проводимо перевірку інформаційної узгодженості СЛІД в ланцюжку. Переконуємося, що всі наступні дії забезпечені інформацією, визначеною попередніми діями, у конкретному випадку операторами ReadLn. Особливу увагу звертаємо на дії, які використовують певну інформацію. Це оператор

```
WriteLn (a:10:4, '*x*x + ', b:10:4, '*x + ', c: 10:4, '= 0: ');
```

та майбутня дія

```
{ Саме рішення квадратного рівняння}.
```

Переконуємося, що на момент виконання значення коефіцієнтів a, b, c вже визначено. Тепер можна зібрати реалізовану частину програми та шляхом її виконання на тестах переконатися, що дії введення та виведення введеної інформації програми виконуються коректно.

```
Program Kvadrat;
{ Програма розв'язання квадратного рівняння
виду a * x * x + Ь * x + c = 0 з довільними значеннями
коефіцієнтів a, Ь, з типу речовий }
Uses
Crt, Dos;
Var
a, b, c: Real; {Коефіцієнти квадратного рівняння}
xl, x2: Real; {Корні квадратного рівняння}
begin
ClrScr; {Очищення екрана}
{Виведення інформації про значення програми}
WriteLn ('Програма розв'язання квадратного рівняння');
Write ('виду a * x * x + b * x + c = 0 з довільними');
Write ('значеннями');
WriteLn ('коефіцієнтів a, b, з типу речовий');
WriteLn
{Введення значень коефіцієнтів a, b, c}
Write ('Вкажіть значення коефіцієнта a =');
ReadLn (a); {Введення a}
Write ('Вкажіть значення коефіцієнта b=');
ReadLn (b); {Введення b}
```

```
Write ('Вкажіть значення коефіцієнта з =');
ReadLn(c); {Введення з}
{ Висновок перевірочно-протокольної інформації про введені значення коефіцієнтів a, Ь, c}
WriteLn;
WriteLn ('Вирішується квадратне рівняння');
Write (a:10:4, '*x*x + ', b:10:4, '*x + ');
WriteLn (c:10:4, ' = 0 : ' );
{ Саме рішення квадратного рівняння }
WriteLn;
WriteLn ('Для завершення програми натисніть ', 'будь-яку клавішу...');
repeat until KeyPressed; { Цикл очікування натискання будь-якої клавіші}
end.
```

Під час складання програми довелося здійснити перенесення частини оператора WriteLn на новий рядок.

Тепер здійснюємо декомпозицію дії «Само розв'язання квадратного рівняння».

Багатоваріантність обчислень передбачає ланцюжок альтернатив. Аналізуючи математичні формули узагальнюючого тесту, табл. 5.3 та склад наборів вихідної інформації, виявляємо, що ланцюжок альтернатив містить чотири альтернативні дії. Рядкам з 1-ї по 3-ю табл. 5.3 відповідає одна дія, оскільки для їх виконання потрібно обчислене значення дискримінанта d. Записуємо коментар попереднього СЛІДЖЕННЯ всього ланцюжка альтернатив, набір вхідної інформації (вихідної інформації - ні) і оформляємо заготівлю операторів ланцюжка альтернатив разом з підлеглими Слідуваннями:

Вхідна інформація: a, b, c

```
{ Саме рішення квадратного рівняння }
if
then
begin
{Продовження рішення з обчисленням дискримінанта}
end;
if
then
begin
{ Рішення лінійного рівняння }
end;
if
then
begin
{ Введення повідомлення: лінійне рівняння не має рішення}
WriteLn ("Немає рішення")
end;
if
then
begin
{ Висновок повідомлення: безліч рішень рівняння }
Write ('незліченна безліч рішень на рівні');
WriteLn ('ня (коріння - будь-які числа)');
end;
```

В останній альтернативі один рядок виводиться одним оператором.

Далі відповідно до дій запишемо логічні умови виконання дій. При цьому простим порівнянням перевіряти на рівність значення двох речових змінних не можна. Наприклад, при порівнянні f = g, які вважаються рівними 5, навіть якщо g = 5,00000, в силу округлень при обчисленнях значення f може бути рівним або 4,99999, або 5,00000, або 5,00001. Згідно з цим прикладом шляхом простої перевірки на рівність факт рівності буде встановлений в одному випадку з трьох.

Для надійного порівняння двох дійсних чисел використовують прийом використання нерівності $| f - g | \leq \varepsilon$, де ε — свідомо мале число. Мовою програмування ця нерівність має вигляд

```
Abs (f - g) <= 1e - 6
```

Продовжуємо кодування структури. Дивлячись на дії, записуємо логічні умови виконання дій. Вхідна інформація: a, b, c.

```
{ Саме рішення квадратного рівняння }
if (Abs(a) > 1e - 6)
then
begin
```

```
{Продовження рішення з обчисленням дискримінанта}
end;
if ((Abs (a) <= 1e - 6) and (Abs (b) > 1e - 6))
then
begin
{ Рішення лінійного рівняння }
end;
if ((Abs(a) <= 1e - 6) and (Abs(b) <= 1e - 6 and (Abs(c) > 1e - 6))
then
begin
{ Висновок повідомлення: лінійне рівняння не має рішення}
WriteLn ('Немає рішення');
end;
if ((Abs(a) <= 1e - 6) and (Abs(b) <= 1e - 6 and (Abs(c) <= 1e - 6))
then
begin
{ Висновок повідомлення: безліч рішень рівняння }
Write ('незліченна безліч рішень на рівні');
WriteLn ('ня (коріння - будь-які числа)');
end;
```

Здійснимо складання програми, що вийшла. При складанні видалимо надлишкові коментарі та надмірні операторні дужки begin - end, що охоплюють лише один оператор. Випробуємо отриману програму на тестах a = 0, b = 0, c = 0 a = 0, b = 0, c = 2. Зібраний варіант програми:

```
Program Kvadrat;
{ Програма розв'язання квадратного рівняння
виду a * x * x + b * x + c = 0 з довільними значеннями
коефіцієнтів a, b, c типу речовий }
Uses
Crt, Dos;
Var
a, b, c: Real; {Коефіцієнти квадратного рівняння}
xl, x2: Real; {Корні квадратного рівняння}
begin
ClrScr; { Очищення екрана }
{Виведення інформації про призначення програми}
WriteLn ('Програма розв'язання квадратного рівняння');
Write ("виду a * x * x + b * x + c = 0 з довільними");
Write ('значеннями');
WriteLn ('коефіцієнтів a, b, c типу речовий');
WriteLn;
{Введення значень коефіцієнтів a, b, c};
Write ('Вкажіть значення коефіцієнта a =');
ReadLn(a); {Введення a}
Write ('Вкажіть значення коефіцієнта b=');
ReadLn(b); {Введення b}
Write ('Вкажіть значення коефіцієнта з =');
ReadLn(c); {Введення з}
{ Висновок перевірочно-протокольної інформації
про введені значення коефіцієнтів a, b, c}
WriteLn;
WriteLn ('Вирішується квадратне рівняння');
Write (a:10:4, '*x*x + ', b:10:4, '*x + ');
WriteLn(з:10:4, '= 0:');
{ Саме рішення квадратного рівняння }
if (Abs (a) > 1e - 6)
then
begin
{Продовження рішення з обчисленням дискримінанта}
end;
if ((Abs(a) <= 1e - 6) and (Abs(b) > 1e - 6))
then
begin
{ Рішення лінійного рівняння }
end;
if ((Abs(a) <= 1e - 6) and (Abs(b) <= 1e - 6) and (Abs(c) > 1e - 6))
```

```
then
WriteLn ('Немає рішення');
if ((Abs(a) <= 1e - 6) and (Abs(b) <= 1e - 6 and (Abs(c) <= 1e - 6))
then
begin
Write ('незліченна безліч рішень на рівні');
WriteLn ('ня (коріння - будь-які числа)');
end;
WriteLn;
Write ('Для завершення програми натисніть');
WriteLn ("будь-яку клавішу ...");
repeat until KeyPressed; { Цикл очікування натискання будь-якої клавіші }
end.
```

Декомпозуємо дію «Рішення лінійного рівняння». Ця дія представляє ланцюжок із двох елементарних операторів. Виконаємо перевірку інформаційної узгодженості дій:

Вхідна інформація: b, c.

```
{ Рішення лінійного рівняння }
x1 := -c/b;
WriteLn ('рівняння лінійне x = ', x1: 10: 4);
```

Декомпозуємо дію "Продовження рішення з обчисленням дискримінанта". Дана дія представляє ланцюжок Слідкувань з двох Слідкувань.

Вхідна інформація: a, Ь, c

```
{Продовження рішення з обчисленням дискримінанта}
{Обчислення дискримінанта квадратного рівняння}
d := Sqr (b) - 4.0 * a * c; { Рішення рівняння }
```

Змінна d у нас не описана, тому до секції Var необхідно додати рядок опису:

```
d: Real; { Значення дискримінанта }
```

Декомпозуємо дію «Рішення рівняння». Відповідно до табл. 5.3 ця дія представляє ланцюжок альтернатив з трьох альтернатив в ланцюжку. Здійснивши деталізацію цих альтернатив у встановленому порядку, отримаємо:

Вхідна інформація: a, b, c, d.

```
{ Рішення рівняння }
if d > 1e-6
then
begin
{Розрахунок двох різних коренів}
end;
if ((d >= -1e-6) and (d <= 1e-6))
then
begin
{Розрахунок двох рівних коренів}
WriteLn ('два рівні корені x = ', (-b)/(2.0 * a) :10:4);
end;
if d < -1e-6 then begin
{Виведення напису: рівняння не має рішення}
WriteLn ('рівняння немає рішення');
end;
```

Декомпозуємо дію «Розрахунок двох різних коренів». Ця дія представляє ланцюжок із трьох елементарних операторів. Виконаємо перевірку інформаційної узгодженості дій:

Вхідна інформація: a, b, c, d

```
{Розрахунок двох різних коренів}
x1 := ((-b) - Sqrt (d))/(2.0*a);
x2 := ((-b) + Sqrt (d))/(2.0*a);
Write ('два різні корені x1 = ', x1:10:4);
WriteLn ('x2 =', x2:10:4);
```

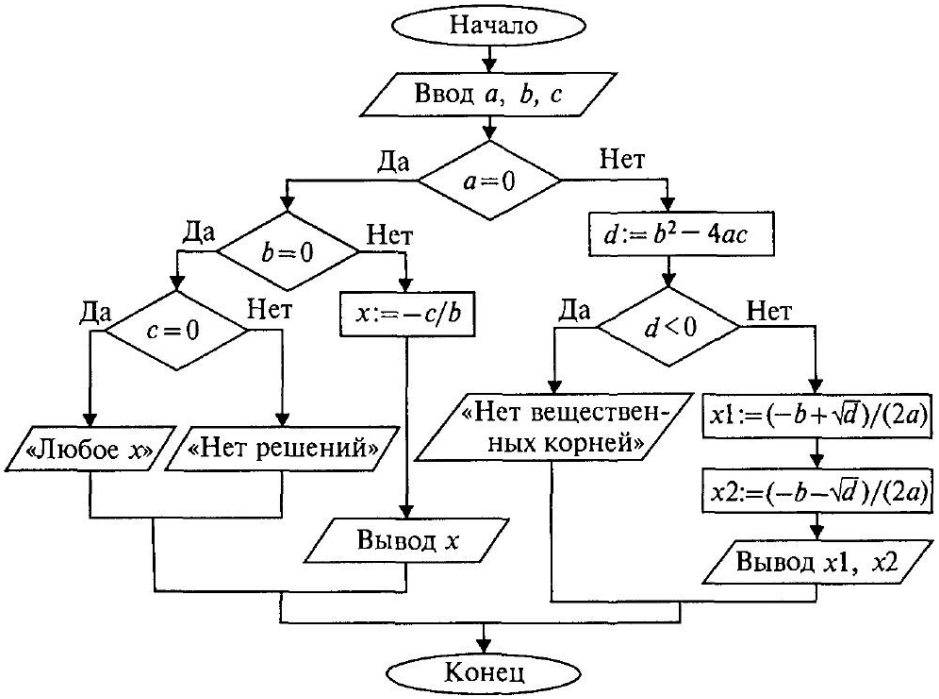
Тут виведення одного рядка здійснено двома операторами. Здійснимо складання всієї програми, вилучивши надлишкові коментарі та надлишкові операторні дужки begin - end, що охоплюють лише один оператор. Випробуємо отриману програму на всіх заздалегідь підготовлених тестах. Зібраний варіант програми:

```
Program Kvadrat;
{ Програма розв'язання квадратного рівняння
виду a * x * x + b * x + c = 0 з довільними значеннями
```

```
коефіцієнтів a, b, c типу речовий }
Uses
Crt, Dos;
Var
a, b, c: Real; {Коефіцієнти квадратного рівняння}
xl, x2: Real; {Корні квадратного рівняння}
dl: Real; {Значення дискримінанта}
begin
ClrScr; { Очищення екрана }
{Виведення інформації про призначення програми}
WriteLn ('Програма розв'язання квадратного рівняння');
WriteLn ("виду a * x * x + b * x + c = 0 з довільними", "значеннями");
WriteLn ('коефіцієнтів a, b, c типу', 'речовий');
WriteLn
{Введення значень коефіцієнтів a, b, c}
Write ('Вкажіть значення коефіцієнта a =');
ReadLn(a); { Введення a }
Write ('Вкажіть значення коефіцієнта b=');
ReadLn(b); {Введення b}
Write ('Вкажіть значення коефіцієнта з =');
ReadLn(c); { Введення з }
{ Висновок перевірочно-протокольної інформації
про введені значення коефіцієнтів a, b, c}
WriteLn;
WriteLn ('Вирішується квадратне рівняння');
WriteLn (a:10:4, '*x*x + ', b:10:4, '*x + ', з:10:4, '= 0:');
{ Саме рішення квадратного рівняння }
if (Abs (a) > 1e-6)
then
begin
{Продовження рішення з обчисленням дискримінанта}
{Обчислення дискримінанта квадратного рівняння}
d := Sqr(b) - 4.0 * a * c;
{ Рішення рівняння }
if d > 1e-6
then
begin
{Розрахунок двох різних коренів}
x1 := (-b) - Sqrt(d)/(2.0*a);
x2: = (-b) + Sqrt (d) / (2.0 * a);
Write ('два різні корені x1 = ', x1:10:4);
WriteLn ( 'x2 = ', x2: 10:4);
end;
if ((d >= -1e-6) and (d <= 1e-6))
then
WriteLn ('два рівні корені x = ', (-b)/(2.0*a):10:4);
if d <-1e-6 then
WriteLn ('рівняння немає рішення');
end;
if ((Abs(a) <= 1e-6) and (Abs(b) > 1e-6))
then
begin
{ Рішення лінійного рівняння }
x1 := -c/b;
WriteLn ('рівняння лінійне x = ', x1:10:4);
end;
if ((Abs(a) <= 1e-6) and (Abs(b) <= 1e-6 and (Abs(c) > 1e-6))
then
WriteLn ('Немає рішення');
if ((Abs(a) <= 1e-6 and (Abs(b) <= 1e-6 and (Abs(c) <= 1e-6)))
then
begin
Write ("Безліч рішень",
'рівні');
WriteLn ('ня (коріння - будь-які числа)');
end;
```

```
WriteLn;
Write ('Для завершення програми натисніть');
WriteLn ('будь-яку клавішу ...');
repeat until KeyPressed; { Цикл очікування натискання будь-якої клавіші }
end.
```

Для інтересу можна навести блок-схему:



5.10. ПРИКЛАД ВИКОНАННЯ НАВЧАЛЬНОЇ РОБОТИ «РОЗРОБКА АЛГОРИТМА ПРИМНОЖЕННЯ»

Як приклад наводиться навчальна робота, виконана одним із учнів. Роботу було оформлено на окремих аркушах формату А4. Курсивом виділено пояснення авторів підручника, які були додатково ними внесені до тексту роботи.

Сторінка 1(без нумерації) є титульний лист з найменуванням: «ЗАВАННЯ НА СКЛАДАННЯ СТРУКТУРОВАНОГО АЛГОРИТМУ».

Сторінка 2містить постановку завдання та набір тестів, складених до розробки алгоритму процесу.

КРОК 1. ПОСТАНОВКА ЗАВДАННЯ

Скласти алгоритм множення двох позитивних чисел із довільною (до ста) кількістю цифр. Цифри співмножників та результату повинні знаходитися в одновимірних масивах. Розрядність результату має перевищувати 100 цифр.

Крок 2. НАБІР ТЕСТІВ, СКЛАДЕНИХ ДО РОЗРОБКИ АЛГОРИТМУ ПРОЦЕСУ

Нехай гранична розрядність співмножників дорівнює трьом цифрам, а результату чотирьом. Аналогічно наведеному зразку множення чисел 391*56 = 21896 (переповнення) було складено тести: 23*132 = 3036; 111 * 11 = 1221; 999 * 99 = 98901 (переповнення); 00 * 000 = 0; 1 * 0 = 0.

```
   3 9 0
 ×
     5 6
 ---------
 0 2 2 3 5 4 0 0
+
 0 1 1 9 4 5 0 0
 ---------------
 0 2 1 1 0 8 4 0 0
```

Алгоритм множення зазвичай вивчався у молодших класах школи з маршрутного опису процесу рахунки. Через теоретичну кількість маршрутів більшість зі шкільної лави не знає процесу множення при нульових співмножниках!

Сторінка 3містить результати аналізу вихідної та вхідної інформації обчислювального процесу зі структурами даних. Раціональність вибраної структури даних значною мірою визначає раціональність алгоритму.

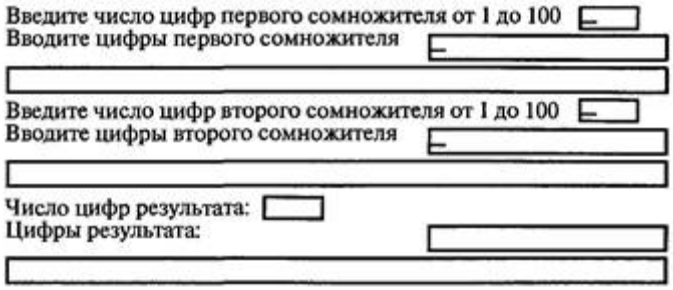
Крок 3. АНАЛІЗ ВИХІДНОЇ ТА ВХІДНОЇ ІНФОРМАЦІЇ ВИЧИСЛЮВАЛЬНОГО ПРОЦЕСУ

Аналіз вихідний та вхідний інформації починається з розгляду моделі «чорного» ящика, показаної на рис. 5.3.

```
Program MultNumbers;
{Розрахунок добутку двох чисел}
uses
Crt;
const
Digits = 100; {Число цифр у числах}
type
TNumber = record
D: array[1..Digits] of Byte;
{BD[1] знаходиться молодший розряд числа}
N: word; {Число розрядів у числі від 1 до Digits}
end;
var
C1: TNumber; {Перший співмножник}
C2: TNumber; {Другий співмножник}
R: TNumber; {Результат множення}
Error: boolean; {True - помилка переповнення}
```

Макет екрану з рядками діалогу програми наведено на рис. 5.17. Замість трьох останніх рядків можливий висновок: "Помилка переповнення".

Сторінка 4 містить наочне зображення процесу перетворення вхідних даних узагальнюючого тесту або тестів у вихідні дані з усіма внутрішніми даними та/або трасу виконання узагальнюючого тесту або тестів. Узагальнюючий тест чи тести складаються з урахуванням тестів сторінки 2 і за мінімальної кількості тестів охоплює всі маршрути процесу обчислень. Наочність зображення змін усіх даних сприяє спрощенню процесу розробки алгоритму. Раціональність вибраної структури даних значною мірою визначає раціональність алгоритму.



*Мал. 5.17.*Макет екрану з рядками діалогу програми

Крок 4. НАГЛЯДНЕ ЗОБРАЖЕННЯ ПРОЦЕСУ ПЕРЕТВОРЕННЯ ВХІДНИХ ДАНИХ УЗАГАЛЬНОГО ТЕСТА У ВИХІДНІ

```
      C1.N = [3] [2] [1] C1.D    i
             3   9   0
        x C2.N = [2] [1] C2.D    j
                 5   6
      ------------------------
  +        2  ²3  ⁵4  00
       1  ¹9  ⁴5  °0
      ------------------------
        2  ¹1  °8  °4  °0
R.N = [5] [4] [3] [2] [1] R.D
```

Вводимо опис нових внутрішніх змінних:

```
var {Робочі змінні}
i, j: word;
```

На перший погляд, здається, що необхідно ввести змінну кількість проміжних масивів (залежно кількості цифр другого співмножника) для запам'ятовування продукту множення першого співмножника на чергову цифру другого співмножника. Однак уважний аналіз структур даних дозволяє знайти інший спосіб виконання розрахунків. При цьому способі спочатку обнулюється результат. Далі результат послідовно збільшується на зрушений на розряд ліворуч продукт множення першого співмножника на чергову цифру другого співмножника. Переоформляємо наочне зображення процесу перетворення вхідних даних узагальнюючого тесту.

```
R.D =              0      R.N = 0
        [1] +             Итерация 1.
    390 x 6  =  2340      P
R.D =           2340      R.N = 4
R.D =           2340      R.N = 4
        [2] +             Итерация 1.
    390 x 5  =  1950      P
R.D =          21840      R.N = 5
```

Новые переменные:

```
var
  p     : word;  {Значение числа переноса при
                  умножении C1.D на очередную
                  цифру C2.D}
```

На сторінці 4 або наступній іноді корисно помістити опис алгоритму у звичайному неструктурованому розумінні.

Кожна наступна сторінка *містить результати процесу деталізації чергової виділеної від загального до приватного структури.*

Крок 5. НАСЛІДНІСТЬ ДЕТАЛІЗАЦІЇ АЛГОРИТМУ

Крок 5.1. Результати деталізації СЛІД «Вся програма»

Слідування «Вся програма» деталізується ланцюжком Слідування:

```
begin
ClrScr; {Очищення екрана}
{Введення коректного значення числа цифр першого співмножника}
C1.N
Write('Вводите цифри першого співмножника');
{Введення цифр першого співмножника у порядку від C1.D[C1.N] до C1.D[1]}
C1.D
WriteLn;
{Введення коректного значення числа цифр другого співмножника}
C2.
Write('Вводьте цифри другого співмножника');
{Введення цифр другого співмножника у порядку від C2.D[C2.N] до C2.D[1]}
WriteLn;
{Розрахунок твору співмножників}
RD RN Error
{Усунення провідних нулів}
```

```
WriteLn;
{Виведення результату твору}
WriteLn;
end.
```

Без відступів показана вхідна та вихідна інформація структур, яка використовувалась при перевірці інформаційної узгодженості СЛІД в ланцюжку СЛІД.

СЛІД «Усунення провідних нулів» необхідно при використанні співмножника, що складається з декількох нулів.

Крок 5.2. Деталізація СЛІД «Введення коректного значення числа цифр першого співмножника»

СЛІД «Введення коректного значення числа цифр першого співмножника» декомпозується циклом:

```
{Введення коректного значення числа цифр першого співмножника}
repeat
Write('Введіть число цифр першого співмножника');
Write(' від 1 до ', Digits, '');
ReadLn(C1.N);
until ((C1.N >= 1) and (C1.N <= Digits));
```

Цикл тестований трьома тестами: C1.N=1; C1.N=3; C1.N=Digits.

Аналогічно декомпозується процес "Введення коректного значення числа цифр другого співмножника".

Крок 5.3. Деталізація СЛІД «Введення цифр першого співмножника в порядку від C1.D[C1.N] до C1.D[1]

СЛІД «Введення цифр першого співмножника в порядку від C1.D[C1.N] до C1.D[1] декомпозується циклом:

```
{Введення цифр першого співмножника у порядку від C1.D[C1.N] до C1.D[1]}
for i := C1.N downto 1 do begin
{До введення символу цифри}
repeat
ch: = ReadKey; {Читання символу клавіатури}
Val(ch, C1.D[i], InCode); {Перетворення на значення}
until(InCode = 0);
Write(ch);
end;
```

Опис нових змінних:

```
var {Робочі змінні}
InCode: слово;
ch: Char;
```

Незважаючи на те, що тут, порушуючи правила, деталізовано відразу два цикли, у тестуванні немає необхідності.

Аналогічно декомпозується процес «Введення цифр другого співмножника у порядку від C2.D[C2.N] до C2.D[1].

Крок 5.4. Деталізація СЛІД «Висновок результату твору»

СЛІД «Висновок результату твору» декомпозується АЛЬТЕРНАТИВОЮ - РОЗВИТОК З ДВОМИ ДІЯМИ:

```
{Виведення результату твору}
if ERROR
then
WriteLn(Помилка переповнення)
else
begin
{Виведення продукту множення}
end;
```

Тести: ERROR = True; ERROR = False.

Крок 5.5. Деталізація СЛІД «Виведення продукту множення»

СЛІД «Виведення продукту множення» декомпозується циклом:

```
{Виведення продукту множення}
for i := RN downto 1 do
Write(RD[i]);
```

У тестуванні немає потреби.

Крок 5.6. Деталізація СЛІДЖЕННЯ «Усунення провідних нулів»

СЛІД «Усунення провідних нулів» декомпозується циклом:

```
{Усунення провідних нулів}
```

```
while ((RN > 1) and (RD[i] = 0)) do
Dec(RN); {RN := RN - 1}
```

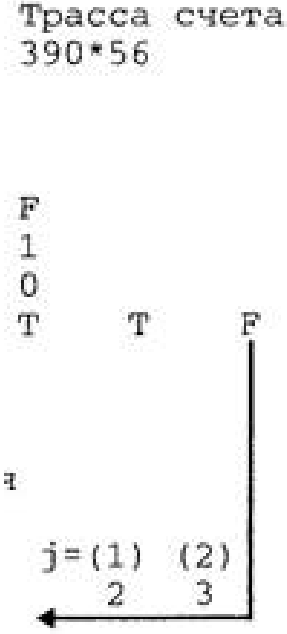
У тестуванні немає потреби.

Крок 5.7. Деталізація СЛІД «Розрахунок твору співмножників»

СЛІД «Розрахунок твору співмножників» декомпозується циклом:

Вхід: C1, C2.

```
{Розрахунок твору співмножників}
{Цикл визначає номер j чергової цифри другого співмножника}
ERROR: = False;
j: = 1;
RD [1]: = 0;
while ((j <= C2.N) and
(not(ERROR))) do
begin
{Збільшення результату на зрушений продукт множення першого співмножника на j-у цифру
другого співмножника}
Inc(j); {j:=j+1}
end;
```



Вихід: RD, RN, ERROR

Структура тестувалась на тестах: 390*56; 390 * 56, але при Digits = 5; 0 * 0 при C1.N = 0; 1*0 при C1.N = 1 та інших тестах.

Крок 5.8. Деталізація СЛІДЖЕННЯ «Збільшення результату на зрушений продукт множення першого співмножника на j-у цифру другого співмножника

СЛЕДЖЕННЯ деталізується циклом:

Вхід: C1, C2.

{Збільшення результату на зрушений продукт множення першого співмножника на j-у цифру другого співмножника}

```
p: = 0;
i := 0; {Номер цифри першого співмножника}
while(((i < C1.N) or (p <> 0)) and (not(ERROR))) do
begin
Inc(i);
{Розрахунок чергової цифри результату та цифри перенесення}
end;
```

Вихід: RD, RN, ERROR

Крок 5.9. Деталізація СЛІД «Розрахунок чергової цифри результату та цифри перенесення»

СЛЕДЖЕННЯ деталізується альтернативою:

Вхід: i, j, C1.D[i], C2.D[j], p, RD, Digits.

```
{Розрахунок чергової цифри результату та цифри перенесення}
{Контрольований розрахунок ir — номери чергової цифри результату}
ir := i + j - 1;
if (ir > Digits) then
ERROR := True else
begin
{Зміна довжини результату RN}
if (RN < ir)
then
begin
RN := ir;
RD [ir]: = 0; {Обнулення нової цифри результату}
end;
{Отримання чергової цифри C1D першого співмножника}
if (i <= C1.N) then C1D := C1.D[i] else C1D := 0;
{Зміна чергової цифри результату та p}
RD: = p + RD [ir] + C1D * C2.D [j];
RD[ir]: = RD mod 10;
p := RD div 10;
end;
```

Вихід: RD, RN, ERROR, p.

Опис нових змінних:

```
var
ir, C1D, RD: слово; {Робочі змінні}
```

Крок 6. РЕЗУЛЬТАТИ ЗБІРАННЯ ПРОГРАМИ

```
Program MultNumbers;
{Розрахунок добутку двох чисел}
uses
Crt;
const
Digits = 100; {Число цифр у числах}
type
TNumber = record
D: array[1..Digits] of Byte;
{BD[1] знаходиться молодший розряд числа}
N: word; {Число розрядів у числі від 1 до Digits}
end;
var
C1: TNumber; {Перший співмножник}
C2: TNumber; {Другий співмножник}
R: TNumber; {Результат множення}
Error: boolean; {True - помилка переповнення}
var
p: word; {Значення числа перенесення при множенні C1.D на чергову цифру C2.D}
var {Робочі змінні}
i, j, ir, C1D, RD, InCode: word;
ch: char; begin
ClrScr; {Очищення екрана}
{Введення коректного значення числа цифр першого співмножника}
repeat
Write('Введіть число цифр першого співмножника)
Write('1 до', Digits,'');
ReadLn(C1.N);
until ((C1.N >= 1) and (C1.N <= Digits));
Write('Вводите цифри першого співмножника');
{Введення цифр першого співмножника у порядку від C1.D[C1.N] до C1.D[1]}
for i := C1.N downto 1 do
begin
{До введення символу цифри}
repeat
ch: = ReadKey; {Читання символу клавіатури}
Val(ch, C1.D[i], InCode); {Перетворення на значення}
until(InCode = 0);
Write(ch);
```

```
end;
WriteLn;
{Введення коректного значення числа цифр другого співмножника}
repeat
Write('Введіть число цифр другого співмножника');
Write(' від 1 до ', Digits,' ');
ReadLn(C2.N);
until ((C2.N >= 1) and (C2.N <= Digits));
Write('Вводьте цифри другого співмножника');
{Введення цифр другого співмножника у порядку від C2.D[C2.N] до C2.D[1]}
for i := C2.N downto 1 do
begin
{До введення символу цифри}
repeat
ch: = ReadKey; {Читання символу клавіатури}
Val(ch, C2.D[i], InCode); {Перетворення на значення}
until(InCode = 0);
Write(ch);
end;
WriteLn;
{Розрахунок твору співмножників}
{Цикл визначає номер j чергової цифри другого співмножника}
ERROR: = False;
j: = 1;
RD [1]: = 0;
while ((j <= C2.N) and (not(ERROR))) do
begin
{Збільшення результату на зрушений продукт множення першого співмножника на j-у цифру
другого співмножника}
P: = 0;
i := 0; {Номер цифри першого співмножника}
while(((i < C1.N) or (p <> 0)) and (not(ERROR))) do
begin
Inc(i);
{Розрахунок чергової цифри результату та цифри перенесення}
{Контрольований розрахунок ir — номери чергової цифри результату}
ir: = i + j - 1;
if (ir > Digits) then
ERROR := True
else
begin
{Зміна довжини результату RN}
if (RN < ir)
then
begin
RN := ir;
RD [ir]: = 0; {Обнулення нової цифри результату}
end;
{Отримання чергової цифри C1D першого співмножника}
if (i <= C1.N)
then
C1D := C1.D[i]
else
C1D: = 0;
{Зміна чергової цифри результату та p}
RD: = p + RD [ir] + C1D * C2.D [j];
RD[ir]: = RD mod 10;
p := RD div 10;
end;
end;
Inc(j); {j:=j+1}
end;
{Усунення провідних нулів}
while ((RN > 1) and (RD[i] = 0)) do
Dec(RN); {RN := RN - 1}
WriteLn;
```

```
{Виведення результату твору}
if ERROR
then
WriteLn('Помилка переповнення')
else
begin
{Виведення продукту множення}
for i := RN downto 1 do
Write(RD[i]);
end;
WriteLn;
end.
```

Після закінчення складання програми має сенс ще раз відредагувати коментарі із вилученням «зайвих» коментарів.

5.11. ПРИКЛАД ЗАСТОСУВАННЯ ПРОЕКТНОЇ ПРОЦЕДУРИ ДЛЯ КОДИРУВАННЯ ПРОГРАМИ ДРУКУ КАЛЕНДАРЯ НА ПРИНТЕРІ

Нехай потрібно розробити програму друку календаря заданого року на матричному принтері. Обмежимося роками після 1917 р. Матричний принтер може друкувати інформацію послідовно один рядок за другим.

Нижче наведено розроблений макет друку вихідної інформації. З аналізу макета стало очевидно, що вхідною інформацією програми є лише рік календаря, що виводиться на друк. Розробляємо макет екрану діалогу програми.

Приступаємо до розробки оперативної структури даних програми. Що потрібно для друку календаря року? Звичайно, можна створити статичну матрицю, яка містить усі символи календаря, або статичний вектор усіх рядків макету календаря. Спробуємо обійтися без великих масивів, формуючи послідовно черговий рядок інформації.

Що бачимо? Інформація дат місяця представлена на макеті сімома рядками та шістьма колонками дат місяців. Що потрібне для друку дат конкретного місяця? Кількість порожніх клітин дат на початку місяця та кількість днів на місяці. Кількість днів у всіх місяцях року фіксована, за винятком лютого. Кількість днів у лютому залежить від того, чи є рік високосним чи ні. Отже, кількість днів у всіх місяцях року маємо у статичному масиві.

Кількість порожніх клітин дат на початку наступного місяця визначається інформацією попереднього місяця. Цю інформацію також маємо в статичному масиві. Для заповнення статичного масиву кількості порожніх клітин дат на початку місяців року необхідно знати інформацію про кількість порожніх клітин дат на початку січня року друку календаря.

Кількість порожніх клітин дат на початку січня року друку календаря можна визначити виходячи з кількості порожніх клітин дат на початку січня 1917 з урахуванням виявленого з макета наступного факту, який полягає в тому, що кількість порожніх клітин дат на початку січня наступного року більше на 1 у невисокосний рік та на 2 у високосний рік.

```
Програма друку календаря заданого року
Вкажіть рік роздрукованого календаря після 1917 року
■■■■
(Не можу скласти календар ■■■■ року
Щоб завершити програму, натисніть будь-яку клавішу ...)
Чекайте, йде друк...
Щоб завершити програму, натисніть будь-яку клавішу ...
```

2006

	Январь					Февраль					Март				
Пн		2	9	16	23	30	6	13	20	27	6	13	20	27	Пн
Вт		3	10	17	24	31	7	14	21	28	7	14	21	28	Вт
Ср		4	11	18	25	1	8	15	22	1	8	15	22		Ср
Чт		5	12	19	26	2	9	16	23	2	9	16	23		Чт
Пт		6	13	20	27	3	10	17	24	3	10	17	24		Пт
Сб		7	14	21	28	4	11	18	25	4	11	18	25		Сб
Вс	2	8	15	22	29	5	12	19	26	5	12	19	26		Вс
	Апрель					Май					Июнь				
Пн		3	10	17	24	1	8	15	22	29	5	12	19	26	Пн
Вт		4	11	18	25	2	9	16	23	30	6	13	20	27	Вт
Ср		5	12	19	26	3	10	17	24	31	7	14	21	28	Ср
Чт		6	13	20	27	4	11	18	25	1	8	15	22	29	Чт
Пт		7	14	21	28	5	12	19	26	2	9	16	23	30	Пт
Сб	1	8	15	22	29	6	13	20	27	3	10	17	24		Сб
Вс	2	9	16	23	30	7	14	21	28	4	11	18	25		Вс
	Июль					Август					Сентябрь				
Пн		3	10	17	24	31	7	14	21	28	4	11	18	25	Пн
Вт		4	11	18	25	1	8	15	22	29	5	12	19	26	Вт
Ср		5	12	19	26	2	9	16	23	30	6	13	20	27	Ср
Чт		6	13	20	27	3	10	17	24	31	7	14	21	28	Чт
Пт		7	14	21	28	4	11	18	25	1	8	15	22	29	Пт
Сб	1	8	15	22	29	5	12	19	26	2	9	16	23	30	Сб
Вс	2	9	16	23	30	6	13	20	27	3	10	17	24		Вс
	Октябрь					Ноябрь					Декабрь				
Пн		2	9	16	23	30	6	13	20	27	4	11	18	25	Пн
Вт		3	10	17	24	31	7	14	21	28	5	12	19	26	Вт
Ср		4	11	18	25	1	8	15	22	29	6	13	20	27	Ср
Чт		5	12	19	26	2	9	16	23	30	7	14	21	28	Чт
Пт		6	13	20	27	3	10	17	24	1	8	15	22	29	Пт
Сб		7	14	21	28	4	11	18	25	2	9	16	23	30	Сб
Вс	1	8	15	22	29	5	12	19	26	3	10	17	24	31	Вс

Нам потрібно чотири рядки найменувань місяців кварталу та сім рядків найменувань днів тижня. Ця інформація залишається незмінною у процесі виконання програми, тому розміщуємо її у константних масивах Turbo Pascal.

Отже, нам потрібна дія, що багаторазово застосовується: визначення, чи є досліджуваний рік високосним чи ні. Оформляємо цю дію як функцію з результатом логічного типу, що повертається. Алгоритм цієї функції візьмемо зі шкільного підручника з астрономії.
Текст програми друку календаря на матричному принтері наведено нижче. Звіряючись із цим текстом, самостійно закодуйте програму.

```
Program Calendar;
{Програма друку календаря заданого року на матричному принтері.
Рік має бути починаючи з YEARBASE року}
Uses
Crt, Dos;
Const
YEARBASE = 1917; {Початковий базовий рік}
BLANKS1917 = 0; {Кількість порожніх клітин на початку січня базового року}
KVARTALNAME: array [1...4] of String [53] =
( {Константи вкорочені при макетуванні! }
'Січень Лютий Березень',
'Квітень Травень Червень',
' Липень Серпень Вересень',
' Жовтень Листопад Грудень'
);
WEEKDAYNAME: Array [1...7] of String [2] = ('Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Вс');
Var
F: Text; {Файлова змінна}
Year: Word; {Рік календаря, що роздруковується}
Kvartal: Word; {Квартал року}
{Кількість днів у кожному місяці}
MonthsDays: Array [1...12] of Word;
{Кількість порожніх клітин на початку січня року Year}
Balnks: Word;
{Кількість порожніх клітин на початку кожного місяця}
BlanksDaus: Array [1...12] of Word;
```

```
idW: Word; {Номер дня тижня}
iMonth: Word; {Номер місяця на рік}
iKvartalMonth: Word; {Номер місяця у кварталі}
iCol: Word; {Номер колонки у місяці кварталу}
iCell: Word; {Номер клітини у місяці кварталу}
i: Integer;
Function Vys (Year: Word): Boolean;
{Функція повертає True у разі високосності року Year}
begin
Vys: = False;
if (((Year mod 4 = 0) and (Year mod 100<>0))
or (Year mod 1000 = 0))
then
Vys: = True;
end; {Vys}
begin { Основна програма }
ClrScr; {Очищення екрана}
Write ('Програма друку календаря');
WriteLn ('заданого року');
Write ('Вкажіть рік роздруковуваного');
WriteLn ('календаря після', YEARBASE: 5, 'року');
ReadLn (Year);
{Контроль запровадженого року}
if Year < YEARBASE then
begin
{Аварійне завершення програми}
Write ('Не можу скласти календар');
WriteLn (Year: 5, 'року');
Write ('Для завершення програми');
WriteLn ('натисніть будь-яку клавішу...');
repeat until KeyPressed;
Halt (1);
end;
WriteLn ('Чекаєте, йде друк...');
Assign (F, 'PRN');
Rewrite (F);
{Друк календаря на принтері}
{Частина прогалин у наступному рядку була вилучена!}
WriteLn (F, '', Year);
{Підготовка інформації}
{Визначення кількості порожніх клітин у січні року Year}
Blanks: = BLANKS1917;
i := YEARBASE;
while (I Year) do begin
{Збільшення Blanks}
Inc (Blanks); {У будь-який рік плюс 1}
if Vys (i)
then
Inc (Blanks); {Минулий рік високосний, +2}
{Коригування Blanks}
if (Blanks >= 7) then Blanks := Blanks - 7;
Inc (i); {Поточний рік}
end;
{Визначення кількості днів у кожному місяці}
for i := 1 to 12 do
MonthsDays [i]: = 31;
MonthsDays [4]: = 30;
MonthsDays [6]: = 30;
MonthsDays [9]: = 30;
MonthsDays [11]: = 30;
MonthsDays [2]: = 28;
if Vys (Year) then MonthsDays [2] : = 29;
{Визначення кількості порожніх клітин на початку кожного місяця}
BlanksDays [1] := Blanks;
for i : = 2 to 12 do
if BlanksDays [i - 1] + MonthsDays [i - 1] < 35
```

```
then
BlanksDays [i] := BlanksDays [i - 1] + MonthsDays [i - 1] - 28
else
BlanksDays [i]: = BlanksDays [i - 1] + MonthsDays [i - 1] - 35;
{Завдання номерів кварталів}
{Друк тіла календаря}
for Kvartal := 1 to 4 do begin
{Друк найменування кварталу}
WriteLn (F, KVARTALNAME [Kvartal]);
{Друк дат кварталу}
{Завдання номера дня тижня}
for iDW := 1 to 7 do
begin
{Друк найменування днів тижнів}
Write (f, WEEKDAYNAME [iDW];
{Друк трьох місяців дат кварталу}
for iKvartalMonth : = 1 to 3 do begin
{Розрахунок номер місяця у кварталі}
iMonth: = (Kvartal - 1) * 3 + iKvartalMonth;
{Друк шести колонок дат дня тижня кварталу}
for iCol := 1 to 6 do begin
iCell := (iCol - 1)*7 + iDW;
if ((iCell > BlanksDays [iMonth]) and (iCell <= BlanksDays [iMonth] + MonthsDays
[iMonth])))
then
Write (F, iCell - BlanksDays [iMonth] : 3)
else
Write (F, '');
end;
end;
{Друк назви днів тижнів}
WriteLn (F, WEEKDAYNAME [iDW];
end;
end;
Close (F);
Write ('Для завершення програми');
WriteLn ('натисніть будь-яку клавішу...');
Repeat until KeyPressed;
end.
```

ВИСНОВКИ

• З появою ЕОМ актуальним став пошук способів опису обчислювальних алгоритмів. У 60-х роках вже застосовувалися два способи опису алгоритмів: словесний покроковий та графічний у вигляді схем алгоритмів (жаргонно: блок-схем алгоритмів).

• Згідно з сучасними технологіями програмування, описи алгоритмів у словесно покроковій та графічній формах у вигляді схем алгоритмів практично не використовуються. Їх замінили самодокументовані тексти, що складаються із стандартних структур кодування.

• Хорошим функціональним описом є безпомилковий опис, однозначний для читача, короткий, суть якого розуміється швидко. Згідно з проектною процедурою, хороший функціональний опис складається від загального до приватного з використанням особливих конструкцій пропозицій типових елементів (типових структур або просто структур).

• Будь-які алгоритми або єврористи мають складатися лише зі стандартних структур. Кожна стандартна структура має один інформаційний вхід і один інформаційний вихід. Використання інших (нестандартних) структур призводить або до подовження опису, або до неможливості тестування (через неможливо великого обсягу необхідних тестів), або до втрати зрозумілості.

• При розробці єврористів вхідна, проміжна та вихідна інформації зазвичай характеризуються найменуваннями предметів, їх станом, місцем розташування та часом.

• При розробці алгоритмів програм вхідна, проміжна та вихідна інформації характеризуються іменами та набором значень як простих, так і структурованих змінних (записів, масивів).

• Спочатку алгоритм або евроритм повинен представляти одну типову структуру СЛІД (одна дія зі змістом «виконати всі дії програми»). Далі до досягнення елементарних дій (елементарних операторів мови програмування або елементарних операцій) окремі структури Слідування декомпозуються з

дотриманням принципу від загального до приватного однієї з трьох стандартних структур: Ланцюжок Слідкувань; Ланцюжок АЛЬТЕРНАТИВ; ПОВТОРЕННЯ.
• Перехід на нову мову програмування рекомендується розпочинати з розробки стандарту кодування типових обчислювальних структур.
• Алгоритм багато в чому визначається структурою даних.
• Тести – необхідний атрибут розробки алгоритму.
• Узагальнюючий тест або тести – мінімальний набір тестових даних, що охоплюють усі можливі випадки обчислень.
• Алгоритми зі старих книг краще розуміються після їхнього рефакторингу.

КОНТРОЛЬНІ ПИТАННЯ

1. Перерахуйте основні недоліки кожного із способів опису алгоритмів.
2. Що таке функціональний опис?
3. Навіщо призначена проектна процедура розробки функціональних описів?
4. Викладіть вимоги щодо способу мислення користувача проектною процедурою розробки функціональних описів.
5. У яких випадках програмісти можуть використовувати проектну процедуру?
6. У яких випадках програмісти не можуть застосовувати проектну процедуру?
7. Які основні характеристики структур СЛІД, АЛЬТЕРНАТИВ і ПОВТОРЕННЯ?
8. Яким є порядок основних кроків (етапів) виконання проектної процедури?
9. Наведіть приклад опису зовнішніх та внутрішніх даних програми.
10. Навіщо потрібна модель «чорної скриньки»?
11. Назвіть тести до програмування. Призначення. Зміст. Форми.
12. Назвіть ознаки структури Ланцюжок СЛІД.
13. Назвіть порядок деталізації одиночного СЛІДЖЕННЯ.
14. Назвіть порядок деталізації ланцюжка СЛІД.
15. Назвіть ознаки структур типу АЛЬТЕРНАТИВУ.
16. Назвіть порядок деталізації ланцюжка альтернатив.
17. Запишіть приклад кодування альтернатив з однією та двома діями.
18. Запишіть приклад кодування альтернатив з трьома та більше діями.
19. Назвіть ознаки структури ПОВТОРЕННЯ.
20. Назвіть порядок деталізації структур універсальних циклів ДО та ПОКИ.
21. Запишіть приклад кодування структури НЕУНІВЕРСАЛЬНИЙ ЦИКЛ-ДО.
22. Запишіть приклад кодування структур УНІВЕРСАЛЬНІ ЦИКЛИ ДО І ПОКИ.
23. Як робиться доказ коректності алгоритмів під час виконання проектної процедури.
24. Доповніть опис процесу кипіння води в чайнику наочними малюнками.