

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КИЇВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Кафедра програмних систем і технологій

**КОВАЛЮК Т.В.
ПЕТРІВСЬКИЙ В.Я.**

ОСНОВИ ПРОГРАМУВАННЯ

методичні вказівки

Київ-2026

Рецензенти:

Іларіонов Олег Євгенович – к.т.н., доцент, завідувач кафедри інтелектуальних технологій Київського національного університету імені Тараса Шевченка;

Баклан Ігор Всеволодович – к.т.н., доцент, доцент кафедри інформатики та програмної інженерії Київського політехнічного інституту імені Ігоря Сікорського.

Рекомендовано до публікації кафедрою програмних систем і технологій, протокол №15 від «08» квітня 2026 р.

Рекомендовано до публікації Вченою радою факультету інформаційних технологій, протокол № 14 від «26» червня 2026 р.

**КОВАЛЮК ТЕТЯНА ВОЛОДИМИРІВНА,
ПЕТРІВСЬКИЙ ВОЛОДИМИР ЯРОСЛАВОВИЧ**

Основи програмування [Електронний ресурс]: методичні вказівки. Кафедра програмних систем і технологій / Ковалюк Т.В., Петрівський В.Я. – К. : КНУ імені Тараса Шевченка, 2026. – 204 с.

Методичні вказівки призначені для здобувачів вищої освіти першого курсу спеціальності «Інженерія програмного забезпечення» освітньої програми «Інженерія програмного забезпечення». У виданні реалізовано проєктно-орієнтований інкрементальний підхід до навчання, що передбачає поетапне розширення єдиного програмного проєкту впродовж усього лабораторного практикуму з поступовим ускладненням завдань і систематичною рефлексією результатів. Подано рекомендації до виконання лабораторних робіт з дисципліни «Основи програмування» із використанням мови програмування C++. Особливу увагу приділено формуванню алгоритмічного мислення, навичок розробки, тестування та аналізу якості коду, а також дотриманню принципів чистого коду.

Публікується в авторській редакції.

© Ковалюк Т.В., Петрівський В.Я, 2026 рік

ЗМІСТ

РОЗДІЛ 1 ЗАГАЛЬНІ ПОЛОЖЕННЯ	7
1.1 Проєктно-орієнтований інкрементальний підхід до виконання лабораторних робіт	7
1.1.1 Загальна концепція навчальної методології	7
1.1.2 Принципи проєктно-орієнтованого інкрементального виконання лабораторних робіт	7
1.1.3 Мета та освітній ефект проєктно-орієнтованого інкрементального підходу	7
1.2 Теми проєктів для проєктно-орієнтованого інкрементального навчання	8
1.3 Перелік лабораторних робіт в рамках інкрементальних проєктів	12
РОЗДІЛ 2 ПРИНЦИПИ І СТАНДАРТИ КОДУВАННЯ МОВАМИ C/C++	14
2.1 Принципи написання гарного коду	14
2.2 Інструкції до написання гарного коду під час використання парадигми процедурно-орієнтованого програмування	15
РОЗДІЛ 3 ПРОЄКТИ ТА ЛАБОРАТОРНІ РОБОТИ	18
3.1 Картки проєктів	18
3.1.1 Картка проєкту «Програмна бібліотека математичних обчислень»	18
3.1.2 Картка проєкту «Програмна система статистичного аналізу даних»	19
3.1.3 Картка проєкту «Програмна бібліотека комбінаторних алгоритмів та прикладних задач»	21
3.1.4 Картка проєкту «Програмна бібліотека елементарних алгоритмів обчислювальної геометрії та прикладних задач»	23
3.1.5 Картка проєкту «Програмна система керування книжковим фондом (бібліотека книг)»	25
3.1.6 Картка проєкту «Програмна система обліку студентів та їх успішності»	27
3.1.7 Картка проєкту «Програмна система бронювання і купівлі квитків на події (концерти, фестивалі, вистави, екскурсії, конференції тощо)»	28
3.1.8 Картка проєкту «Програмна система бронювання і купівлі авіаквитків»	30
3.1.9 Картка проєкту «Програмний менеджер особистих фінансів»	31
3.1.10 Картка проєкту «Програмна система для планування бюджету домогосподарства»	32
3.1.11 Картка проєкту «Програмна система для симуляції банківських операцій»	34
3.1.12 Картка проєкту «Програмна система для обробки даних про кандидатів і вакансії (кадрове агентство)»	35

3.1.13 Картка проекту «Програмна система для прогнозування попиту на продукцію»	37
3.1.14 Картка проекту «Програмна система аналітики ІТ ринку праці»	38
3.1.15 Картка проекту «Персональний електронний органайзер студента»	39
3.1.16 Картка проекту «Програмна система «Журнал куратора академічної групи»»	40
3.1.17 Картка проекту «Програмна система моніторингу вебстатистики»	42
3.1.18 Картка проекту «Програмна система «Калькулятор подорожей»»	43
3.1.19 Картка проекту «Програмна система для виявлення вразливих паролів»	44
3.1.20 Картка проекту «Програмна система обліку гуманітарної допомоги під час війни»	46
3.1.21 Картка проекту «Програмна система моделювання тактичних операцій з БпЛА»	47
РОЗДІЛ 4 ІНКРЕМЕНТИ ПРОЄКТУ (ЛАБОРАТОРНІ РОБОТИ)	49
4.1 Лабораторна робота №1. Типи даних. Функції введення-виведення. Обчислення виразів	53
4.1.1 Мета та завдання	53
4.1.2 Теоретичні відомості.....	53
4.1.3 Приклад виконання	59
4.2 Лабораторна робота №2. Розгалужені процеси.....	63
4.2.1 Мета та завдання	63
4.2.2 Теоретичні відомості.....	64
4.2.3 Приклад виконання	65
4.2.4 Контрольні запитання	71
4.3 Лабораторна робота №3. Циклічні обчислювальні процеси.....	71
4.3.1 Мета та завдання	71
4.3.2 Теоретичні відомості.....	71
4.3.3 Приклад виконання	73
4.3.4 Контрольні запитання	80
4.4 Лабораторна робота №4. Рекурентні співвідношення.....	81
4.4.1 Мета та завдання	81
4.4.2 Теоретичні відомості.....	81
4.4.3 Приклад виконання	82
4.4.4 Контрольні запитання	90

4.5 Лабораторна робота №5. Рекурсивні функції	90
4.5.1 Мета та завдання	90
4.5.2 Теоретичні відомості.....	91
4.5.3 Приклад виконання	92
4.5.4 Контрольні запитання	100
4.6 Лабораторна робота №6. Одновимірні масиви	100
4.6.1 Мета та завдання	100
4.6.2 Теоретичні відомості.....	100
4.6.3 Приклад виконання	102
4.6.4 Контрольні запитання	112
4.7 Лабораторна робота №7. Багатовимірні масиви	113
4.7.1 Мета та завдання	113
4.7.2 Теоретичні відомості.....	113
4.7.3 Приклад виконання	114
4.7.4 Контрольні запитання	128
4.8 Лабораторна робота №8. Показчики та посилання. Показчики на функції. Масиви показчиків	129
4.8.1 Мета та завдання	129
4.8.2 Теоретичні відомості.....	129
4.8.3 Приклад виконання	132
4.8.4 Контрольні запитання	143
4.9 Лабораторна робота №9. Обробка рядків в мові C++	143
4.9.1 Мета та завдання	143
4.9.2 Теоретичні відомості.....	144
4.9.3 Приклад виконання	147
4.9.4 Контрольні запитання	158
4.10 Лабораторна робота №10. Структури, масиви структур та об'єднання.....	158
4.10.1 Мета та завдання	158
4.10.2 Теоретичні відомості.....	159
4.10.3 Приклад виконання	160
4.10.4 Контрольні запитання	171
4.11 Лабораторна робота №11. Текстові та бінарні файли	171
4.11.1 Мета та завдання	171

4.11.2 Теоретичні відомості.....	172
Використання класів потоків для роботи з файлами в мові C++	174
4.11.3 Приклад виконання	177
4.11.4Контрольні	запитання
192	
РОЗДІЛ 5 ВИМОГИ ДО ПРОГРАМНОГО КОДУ, ЗАХИСТУ ІНКРЕМЕНТІВ І	
ЗВІТУ (ЛАБОРАТОРНИХ РОБІТ).....	193
5.1. Вимоги до програмного коду	193
5.1.1 Структуризація програми. Принципи процедурного та структурного програмування	193
5.1.2 Дотримання стандартів кодування	194
5.1.3 Коментування програмного коду	195
5.2 Вимоги до звітної документації.....	196
5.2.1 Вміст звіту.....	196
5.3 Критерії оцінювання	198
5.3.1 Якість коду інкременту (оцінювання коду лабораторної роботи)	198
5.3.2 Якість захисту (оцінювання ступеня володіння інтелектуальною власністю на програмний продукт)	199
5.3.3 Якість звіту	200
5.3.4 Своєчасність виконання та подання інкрементів (Deadline).....	202
Резюме. Набір сформованих у здобувачів вищої освіти компетентностей за результатами виконання інкрементального проєкту з основ програмування.....	203
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ.....	204

РОЗДІЛ 1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Проєктно-орієнтований інкрементальний підхід до виконання лабораторних робіт

1.1.1 Загальна концепція навчальної методології

Цикл лабораторних робіт з дисципліни «Основи програмування мовами C/C++» побудовано за проєктно-орієнтованим інкрементальним підходом, що передбачає поетапне розширення та ускладнення одного цілісного програмного проєкту.

На відміну від традиційного підходу, де кожна лабораторна робота виконується ізольовано, у цій методиці кожне наступне завдання є продовженням і розвитком попереднього. Таким чином, студенти не лише засвоюють окремі концепції, а й навчаються інтегрувати їх у єдину програмну систему, наближено до реального процесу розробки програмного забезпечення.

1.1.2 Принципи проєктно-орієнтованого інкрементального виконання лабораторних робіт

- безперервність: кожна лабораторна робота базується на коді, реалізованому у попередній. Пропуск або невиконання попереднього етапу унеможливорює початок наступного;
- кумулятивність: функціональність системи нарощується крок за кроком - від простих обчислень до обробки складних структурованих даних та файлів;
- повторне використання (перевикористання) коду: студенти повинні навчитися ефективно використовувати вже створені модулі (функції, структури, масиви тощо), не дублюючи логіку;
- проєктна спрямованість: усі завдання формують єдиний програмний продукт, який на виході включає набір функцій і структур для виконання обчислень, зберігання даних, їх обробки, виведення результатів і збереження у файлі.

1.1.3 Мета та освітній ефект проєктно-орієнтованого інкрементального підходу

Такий підхід дозволяє:

- сформулювати цілісне уявлення про процес розробки програмного забезпечення;
- поглибити розуміння зв'язків між мовними конструкціями C/C++;
- відпрацювати навички роботи з проєктом: від планування до поступової реалізації;
- навчитися налагоджувати та тестувати програму поетапно, що зменшує складність ймовірних помилок;
- розвинути навички читання, рефакторингу та повторного використання коду.

1.2 Теми проєктів для проєктно-орієнтованого інкрементального навчання

Проєктно-орієнтоване інкрементальне навчання передбачає поетапне опанування матеріалу шляхом розв'язання практичних завдань, які поступово ускладнюються та взаємопов'язані між собою. Кожен етап (інкремент) розширює і доповнює результати попереднього, формуючи у здобувачів освіти комплексне бачення задачі та навички її реалізації в повному обсязі.

Тематика запропонованих проєктів підібрана так, щоб забезпечити:

- логічну послідовність у розвитку професійних компетентностей;
- інтеграцію теоретичних знань та практичних умінь;
- формування досвіду аналізу, проєктування, програмування, тестування та документування.

У цьому підрозділі наведено теми проєктів, що реалізують принцип «від простого до складного», охоплюють різні аспекти предметної області та сприяють закріпленню матеріалу з тем курсу. Такий підхід дозволяє не лише глибше засвоїти інструменти і принципи процедурного та структурного програмування, а й навчитися бачити їх у взаємодії, як складові цілісної програмної системи, опанувавши методологію гнучкого розроблення програмного забезпечення (Agile Software Development).

У таблиці 1.1 подані теми проєктів, які розбиті на 11 етапів (інкрементів). Після завершення кожного етапу студент має працюючу, але неповну версію кінцевої програми, яку можна вдосконалити на наступних етапах, додавши до неї додаткові функціональності. Деталі кожного етапу проєкту подані в розділах з описом завдань відповідних лабораторних робіт.

Таблиця 1.1 – Теми проєктів, цілі та результати виконання проєктів

№ теми	Назва проєкту	Мета і результати виконання проєкту
	Програмна бібліотека математичних обчислень	<i>Мета проєкту:</i> розробити набір функцій для обчислення елементарних і спеціальних математичних функцій та формул. <i>Результат виконання проєкту:</i> консольний застосунок із функціями обчислення арифметичних, тригонометричних, експоненціальних функцій, обробка рядів Тейлора, приклади використання, тести точності, імпорт/експорт даних у файли.
	Програмна система статистичного аналізу даних	<i>Мета проєкту:</i> реалізувати базові розрахунки статистичних величин (середнє арифметичне, медіана, стандартне відхилення, мінімум, максимум) для заданого набору даних. <i>Результат виконання проєкту:</i> консольний застосунок із функціями розрахунку статистичних величин, прикладами їх використання, тести точності, імпорт/експорт даних у файли.
	Програмна бібліотека комбінаторних	<i>Мета проєкту:</i> реалізувати набір алгоритмів для комбінаторних задач (перестановки, комбінації, розміщення).

№ теми	Назва проєкту	Мета і результати виконання проєкту
	алгоритмів та прикладних задач	<i>Результат виконання проєкту:</i> консольний застосунок із функціями реалізації комбінаторних алгоритмів, прикладами їх використання, тести точності, імпорт/експорт даних у файли.
	Програмна бібліотека елементарних алгоритмів обчислювальної геометрії та прикладних задач	<i>Мета проєкту:</i> реалізувати набір базових алгоритмів для роботи з геометричними об'єктами та задачами прикладної геометрії. <i>Результат виконання проєкту:</i> консольний застосунок із функціями реалізації алгоритмів обчислювальної геометрії, прикладами їх використання, тести точності, імпорт/експорт даних у файли.
	Програмна система керування книжковим фондом (бібліотека книг)	<i>Мета проєкту:</i> програмне моделювання роботи бібліотеки: каталог книг, каталог користувачів, облік видач і повернень книжок. <i>Результат виконання проєкту:</i> консольний застосунок із функціями створення та ведення каталогів книг та користувачів, пошуком, обліком видач і повернень книжок, звітами популярності, імпорт/експорт даних у файли.
	Програмна система обліку студентів та їх успішності	<i>Мета проєкту:</i> реалізувати набір алгоритмів для ведення журналу успішності студентів з розрахунком їх рейтингів і відвідувань. <i>Результат виконання проєкту:</i> консольний застосунок із матрицею «студент × предмет», рейтингами, звітами, пошуком, імпорт/експорт даних у файли.
	Програмна система бронювання і купівлі квитків на події (концерти, фестивалі, вистави, екскурсії, конференції тощо)	<i>Мета проєкту:</i> програмне моделювання роботи бібліотеки: каталог книг, каталог користувачів, облік видач і повернень книжок. <i>Результат виконання проєкту:</i> консольний застосунок із функціями створення та обробки каталогу подій, бронюванням, пошуком, підрахунком виручки, збереженням у текстових і бінарних файлах.
	Програмна система бронювання і купівлі авіаквитків (залізничних, автобусних)	<i>Мета проєкту:</i> імітувати бронювання і продаж квитків на транспортні рейси <i>Результат виконання проєкту:</i> консольний застосунок із функціями створення та ведення масивів рейсів, пасажирів, бронювання, пошуку за напрямками, підрахунку заповнюваності та доходу, збереженням у текстових і бінарних файлах.
	Програмний менеджер особистих фінансів	<i>Мета проєкту:</i> облік доходів, витрат та планування особистого бюджету. <i>Результат виконання проєкту:</i> консольний застосунок з категоріями транзакцій, звітами, балансом, прогнозом, імпорт/експорт даних у файли.
	Програмна система для планування бюджету домогосподарства	<i>Мета проєкту:</i> облік доходів, витрат та планування бюджету сім'ї. <i>Результат виконання проєкту:</i> консольний застосунок для визначення доходів, обліку витрат, визначенням

№ теми	Назва проєкту	Мета і результати виконання проєкту
		короткострокових і довгострокових фінансових цілей, розподілом бюджету, контролю фактичних витрат та аналізом відхилень від плану, імпортом/експортом даних у файли.
	Програмна система для симуляції банківських операцій	<i>Мета проєкту:</i> моделювання роботи банку (створення й управління клієнтськими рахунками, виконання банківських операцій). <i>Результат виконання проєкту:</i> консольний застосунок для роботи з рахунками, операцій з коштами, конвертації валют за заданим курсом, аналітики, робота з даними (клієнти, рахунки, транзакції), імпорт/експорт у файли
	Програмна система для обробки даних про кандидатів і вакансії в кадровому агентстві	<i>Мета проєкту:</i> імітація підбору персоналу за заданими параметрами та формування шорт-листів (скорочений пріоритетний список відібраних кандидатів). <i>Результат виконання проєкту:</i> консольний застосунок з масивом кандидатів/вакансій, матричним скорингом (метод оцінювання кандидатів), пошуком, фільтрами, звітами, експортом/імпортом у файли.
	Програма система для прогнозування попиту на продукцію (програмне забезпечення, комп'ютерну техніку тощо)	<i>Мета проєкту:</i> розрахунок прогнозу попиту за історичними даними. <i>Результат виконання проєкту:</i> консольний застосунок для збереження та обробки даних про попит, математичні розрахунки, порівняння фактичних і прогнозних значень, підрахунок середньої похибки прогнозу, фільтри, звіти, експорт/імпорт у файли.
	Програмна система аналітики ІТ ринку праці	<i>Мета проєкту:</i> збір та аналіз даних про вакансії, вимоги і зарплати в ІТ. <i>Результат виконання проєкту:</i> консольний застосунок для збору і збереження даних про вакансії, фільтрації та пошуку вакансій за критеріями, аналізу навичок, що найчастіше зустрічаються, візуалізації аналітики ринку «технологія → кількість вакансій → середня зарплата», розрахунку рейтингів і динаміки попиту, формування звітів.
	Персональний електронний органайзер студента	<i>Мета проєкту:</i> планування навчальних справ, дедлайнів і нотаток. <i>Результат виконання проєкту:</i> консольний застосунок з календарем, нагадуваннями, пошуком подій, збереженням у текстових і бінарних файлах.
	Програмна система «Журнал куратора академічної групи»	<i>Мета проєкту:</i> ведення особових справ студента з обліком та аналізом відвідуваності, успішності і дисциплінарних записів групи. <i>Результат виконання проєкту:</i> консольний застосунок з матрицею «студент × предмет», рейтингами, нотатками куратора, пошуком, імпортом/експортом у файли.

№ теми	Назва проєкту	Мета і результати виконання проєкту
	Програмна система моніторингу вебстатистики	<i>Мета проєкту:</i> збір, зберігання та аналіз даних про відвідуваність вебресурсу (кількість відвідувачів, кількість переглядів сторінок, середній час перебування, джерела трафіку тощо), виявляти тренди та формувати звіти маркетингових стратегій. <i>Результат виконання проєкту:</i> консольний застосунок для збору, збереження та визначення статистичних показників, аналіз джерел трафіку, аналітика та візуалізація у звітах, розрахунок рейтингів сайтів, фільтрація та пошук, звіти та імпорт/експорт у файли.
	Програмна система «Калькулятор подорожей»	<i>Мета проєкту:</i> розрахунок вартості подорожі з урахуванням транспорту, проживання, харчування, розваг та додаткових витрат, а також порівняти кілька варіантів маршруту чи бюджету. <i>Результат виконання проєкту:</i> консольний застосунок для введення та зберігання параметрів подорожі, розрахунку вартості подорожі, порівняння декількох варіантів подорожі за ціною та тривалістю, імпорт/експорт у файли, звіт.
л	Програмна система для виявлення вразливих паролів	<i>Мета проєкту:</i> аналізує паролі користувачів на надійність та виявляє ті, що можуть бути легко зламані, використовуючи словникові атаки та базові алгоритми перевірки стійкості. <i>Результат виконання проєкту:</i> консольний застосунок для читання паролів з файлів, перевірка паролів на відповідність різним критеріям, виявлення відповідності найпоширенішим паролям, обчислення оцінки надійності пароля, звіт, запис результатів перевірки у файл.
	*Програмна система обліку гуманітарної допомоги під час війни	<i>Мета проєкту:</i> планування та фактичний облік вантажів, відправників і отримувачів, формування звітів, виявлення невідповідності плану та факту, імпорт/експорт у файли <i>Результат виконання проєкту:</i> консольний застосунок для введення, редагування та видалення записів про вантажі, пошук і фільтрація даних, сортування та підрахунок статистики допомоги.
	*Програмна система моделювання тактичних операцій з БпЛА	<i>Мета проєкту:</i> симулятор планування та виконання місій БпЛА: опис парку дронів, маршрутизація через точки, оцінка енерговитрат і ризиків, планування черги польотів та підготовка звітів. <i>Результат виконання проєкту:</i> консольний застосунок для створення бази парку БпЛА, місії, карта-грід, по-флу зони, планування маршрутів (DFS + матричні витрати), стратегічні компаратори/планувальники, парсинг текстових наказів, лог телеметрії, збереження/ завантаження у текстовому та бінарному форматах, звіти з тривалості, енергії та успішності роботи дронів.

1.3 Перелік лабораторних робіт в рамках інкрементальних проєктів

У межах навчальної дисципліни студенти виконують комплекс лабораторних робіт, які об'єднані спільною темою інкрементального проєкту. Робота над кожним проєктом здійснюється покроково, згідно з методологією гнучкого розроблення програмного забезпечення (Agile Software Development).

Кожен наступний крок розробки проєкту подається як лабораторна робота і є логічним продовженням попереднього, з додаванням нової функціональності до програми, що розробляється. Такий підхід дозволяє не лише засвоїти окремі теми курсу, а й отримати на завершальному етапі повноцінний програмний продукт із завершеною архітектурою, документованим кодом та тестовими сценаріями.

У таблиці 1.2 подано короткий опис послідовності лабораторних робіт та зміст інкрементальних кроків наскрізного проєкту.

Завдання до лабораторних робіт, що відповідають інкрементальним крокам наскрізного проєкту, описані в розділах, присвячених їхньому виконанню.

Таблиця 1.2 – Перелік лабораторних робіт в рамках інкрементального проєкту

№ лаб. роб. (етапу проєкту)	Назва лабораторної роботи	Зміст інкрементального кроку
1	Типи даних. Операції введення-виведення. Обчислення виразів	Базові математичні функції обчислення виразів, введення параметрів, друк результатів
2	Розгалужені обчислювальні процеси	Розгалуження обчислень залежно від значень введених даних із застосуванням умовних операторів <code>if</code> , <code>switch</code> .
3	Циклічні обчислювальні процеси	Повторення обчислень, обробка серій значень з використанням операторів циклу <code>for</code> , <code>while</code> , <code>do while</code> . Вкладені цикли. Перевірка переповнення комірок оперативної пам'яті при виконанні серії обчислень. Розрахунки із заданою точністю.
4	Рекурентні співвідношення	Реалізація алгоритмів, в яких значення кожного члена послідовності визначається через попередні члени цієї ж послідовності. Степеневі ряди (ряди Тейлора, Маклорена). Циклічні обчислювальні процеси з розгалуженнями.
5	Рекурсивні процеси	Створення рекурсивних функцій для розв'язання задач, що мають повторювану структуру. Визначення глибини рекурсії. Дослідження ефективності рекурсивних процесів. Порівняння ефективності рекурсивної та ітеративної реалізацій.
6	Одновимірні масиви	Обробка векторних даних, накопичення результатів. Використання базових операцій обробки одновимірних масивів. Алгоритми сортування та

		пошуку в масивах. Векторна алгебра, поліноми, многочлени, алгебраїчні рівняння
7	Багатовимірні масиви	Використання базових операцій обробки двовимірних масивів (матриць). Алгоритми матричної алгебри.
8	Показчики, посилання, масиви показчиків, показчики на функції.	Динамічне управління даними, передача функцій як аргументів. Робота з розрідженими матрицями.
9	Обробка рядків	Використання базових операцій обробки рядків на основі бібліотеки функцій <code>string.h</code> . Обробка текстів, поданих масивами рядків.
10	Структури та об'єднання	Групування логічно пов'язаних даних, наприклад, опис об'єктів. Раціональне використання оперативної пам'яті через об'єднання. Обробка масиву структур. показчики та посилання на структури даних.
11	Робота з файлами	Обробка текстових і бінарних файлів. Збереження результатів, зчитування даних з файлів

РОЗДІЛ 2 ПРИНЦИПИ І СТАНДАРТИ КОДУВАННЯ МОВАМИ C/C++

2.1 Принципи написання гарного коду

Якість програмного забезпечення напряду залежить від якості коду. Навіть правильно працюючий алгоритм може бути складним для розуміння, супроводу чи модифікації, якщо він написаний без дотримання базових принципів чистого (гарного) та ефективного коду.

Для того щоб програми були зрозумілими, підтримуваними та надійними, програмісти застосовують набір перевірених підходів і правил, які допомагають уникати зайвої складності, дублювання логіки та непередбачуваної поведінки. Ці принципи універсальні: вони однаково корисні і в процедурному, і в об'єктно-орієнтованому програмуванні.

У таблиці 2.1 наведено основні принципи, коротке пояснення їх суті. Такі орієнтири допомагають студентам писати код, який буде зрозумілий не лише їм самим зараз, але й іншим розробникам у майбутньому. Деталі дотримання принципів чистого коду (гарного стилю програмування) подані далі.

Таблиця 2.1 – Принципи написання чистого коду

Категорія	Назва (eng)	Переклад (укр)	Семантика	Приклад коду
Культура розробки	Clean Code	Чистий код	Код повинен бути читабельним: мнемонічні ідентифікатори, короткі функції, відсутність «магічних чисел», логічна структура, коментарі.	Використання назви <code>calculateArea(int radius)</code> замість <code>func1(int r)</code> .
	Boy Scout Rule	Правило бойскаута	Завжди залишайте код після себе трохи кращим, ніж він був до вас.	Поки виправляємо баг, одразу перейменовуємо незрозумілу змінну <code>x</code> на мнемонічний <code>userCount</code> .
Архітектура	Single Responsibility (SRP)	Єдина відповідальність	Кожна функція чи модуль повинні виконувати лише одну задачу.	Замість однієї функції, що зчитує файл, рахує суму і друкує чек, створити три окремі: <code>readData()</code> , <code>sumValues()</code> , <code>printReceipt()</code> .
	High Cohesion	Висока зв'язність	Елементи всередині однієї функції/модуля мають бути тісно пов'язані.	Функція <code>processVector()</code> працює тільки з вектором, а не намагається паралельно змінити глобальні налаштування екрана.
	Low Coupling	Слабка зв'язаність	Модулі мають бути максимально	Передача параметрів у функцію через аргументи, а

			незалежними. Зміна одного не має ламати інший.	не використання глобальних змінних, які можуть змінитися будь-де.
Простота і мінімалізм реалізації	KISS — Keep It Simple, Stupid	Зроби просто	Не ускладнювати рішення там, де можна обійтись простим. Складність — ворог надійності.	Замість рекурсивної функції з 5 параметрів — простий цикл, що робить те саме за 3 рядки.
	DRY — Don't Repeat Yourself	Не повторюйся	Уникати дублювання. Реалізація логіки, що повторюється, у функції.	Якщо обчислення податку зустрічається в 5 місцях, створити функцію <code>double getTax(double amount)</code> і викликати її всюди.
	YAGNI — You Aren't Gonna Need It	Вам це не знадобиться	Не писати зайвий функціонал. Реалізувати лише те, що потрібно зараз згідно з вимогами.	Не створюйте структуру для збереження історії цін, якщо зараз потрібно лише показати поточну ціну.
Надійність і стійкість коду	Fail Fast	Принцип негайної відмови	Якщо виявлено некоректний стан — негайно сигналізувати про помилку, не продовжувати роботу з хибними даними	На початку функції перевіряємо: <code>if (age < 0) cout << "Error";</code>
	Defensive Programming	Захисне програмування	Код повинен коректно обробляти непередбачувані або некоректні вхідні дані.	Перед діленням перевірка <code>if (divisor == 0)</code> , навіть якщо ви впевнені, що дільник не має бути рівним 0.
	Principle of Least Astonishment	Принцип найменшого здивування	Код має працювати очікувано. Жодних «сюрпризів» у логіці.	Функція <code>getMax()</code> має повертати максимум, а не виводити на консоль повідомлення про помилку і закривати програму.

2.2 Інструкції до написання гарного коду під час використання парадигми процедурно-орієнтованого програмування

Процедурно-орієнтоване програмування – це парадигма програмування, що походить від імперативного програмування, заснованого на концепції викликів процедур.

Процедурно-орієнтована програма, по суті, є послідовністю викликів процедур, складні процедури можна декомпонувати на процедури меншої складності за принципом: одна процедура – один логічний блок [2].

Переваги використання процедурного програмування:

- простота;
- ефективність;
- портативність.

Недоліки використання процедурного програмування:

- обмежена гнучкість і складність розширення коду;
- складності супроводу і модифікації великих програм.

Переваги структурного програмування:

- зручність і легкість для розуміння;
- програми легше читати та вивчати;
- помилки легше знайти;
- програми легше обслуговувати;
- підвищення продуктивності розробки прикладних програм.

Основні правила проєктування функцій:

- функція повинна розділити структуру програми на окремі відносно незалежні блоки;
- одна функція має розв'язувати одну локальну задачу;
- потрібно уникати довгих функцій і глибокої їх вкладеності;
- функція не повинна використовувати параметри без необхідності; функція `main()` є коротким описом усієї програми, є точкою входу до програми і виконує роль диспетчера, що викликає інші функції програми;
- потрібно уникати «ланцюгових» викликів функцій.

Розробка процедурної програми ведеться покроково методом «зверху вниз» за принципом «розділяй і володарюй»

Принцип «розділяй і володарюй» означає поділ великої задачі на менші частини, як наслідок просте вирішення малих підзадач, що в свою чергу породжує вирішення великої задачі. Реалізація принципу «розділяй і володарюй» здійснюється через застосування процедур (функцій).

Принцип простоти означає, що слід уникати непотрібних складнощів під час проєктування та програмування.

Принцип повторного використання коду свідчить про доцільність блоку коду, що повторюється, виносити у процедури (функції) і писати код один раз, а викликати функцію стільки разів, скільки потребує алгоритм програми.

Код можна писати по-різному. Але, є загальноприйняті правила "гарного стилю" програмування [2]. Дані правила охоплюють використання аббревіатур, визначення імен структурних одиниць коду, використання дужок та операторів, тощо. Дотримання даних правил підвищує ефективність сприйняття коду розробниками завдяки стандартизації написання коду, що підвищує ефективність розробки програмного забезпечення як у команді, так і одному розробнику. також слід пам'ятати про використання пам'яті при написанні програм. Це стосується використання локалізації змінних та використання "правильних" типів даних для збереження даних. У невеликих програмах це непомітно, проте у масштабних додатках це сильно впливає на продуктивність роботи програмного забезпечення. Знання та використання правил "гарного коду" підвищує рівень розробника програмного забезпечення.

Усі програми мають бути прокоментовані таким чином, щоб легко описувати мету коду та будь-які алгоритми, які використовуються для досягнення мети.

Зрозумілий код – це естетично гарний, структурований і не перевантажений код, який має єдине форматування, оптимальну структуру та логіку, такий, що легко тестується, легко

розширюється, має чіткі та компактні коментарі щодо логіки чи вимог конкретної функціональності.

Документаційні коментарі описують роботу функцій, які аргументи функція приймає, що функція повертає, які винятки може викидати.

Пояснювальні коментарі використовуються всередині функціональних блоків і пояснюють, що відбувається в конкретних алгоритмах програми.

Основні правила написання коментарів: коментарі не повинні дублювати код; гарні коментарі не виправдовують незрозумілий код; якщо програміст не може написати чіткий коментар, проблема може бути в коді; якщо код скопійований, слід надати посилання на оригінальне джерело; коментарі потрібні під час виправлення помилок в коді; використовуйте коментарі, щоб позначити незавершені реалізації.

Потребують коментування такі елементи коду: файли, сигнатура функцій, локальні та глобальні змінні, складні алгоритми обчислення або складні процеси прийняття рішень.

Коментарі мають відповідати на запитання «Що?», «Як?», «Чому?»

Максимально покращити читабельність і логіку коду можна за допомогою форматування коду, зокрема дотримання політики фігурних дужок, прийнятих в редакторі коду VS Code, який розміщує фігурні дужки {} під ключовими словами в окремих рядках коду, дотримання політики відступів, табуляцій, пробілів з використанням горизонтальних та вертикальних відступів, застосування круглих дужок () з ключовими словами та функціями, форматування операторів, вирівнювання блоків оголошень.

РОЗДІЛ 3 ПРОЄКТИ ТА ЛАБОРАТОРНІ РОБОТИ

3.1 Картки проектів

3.1.1 Картка проекту «Програмна бібліотека математичних обчислень»

Мета проекту	Розробити набір функцій для обчислення елементарних і спеціальних математичних функцій та формул.
Результат виконання	Консольний застосунок із функціями обчислення арифметичних, тригонометричних, експоненціальних функцій, обробка рядів Тейлора, приклади використання, тести точності, імпорт/експорт даних у файли.
Складність	Середня

Таблиця 3.1 – Інкременти проекту «Програмна бібліотека математичних обчислень»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та введення/виведення	Реалізація базових арифметичних функцій (додавання, віднімання, множення, ділення, степінь). Консольне меню. Ввід параметрів користувачем	Консольна програма, що дозволяє виконувати базові математичні обчислення
2	Розгалужені процеси	Додавання умовного вибору операцій (switch/if). Обробка помилок (ділення на нуль, некоректний ввід). Вибір типу функції (арифметична, тригонометрична)	Інтерактивне меню з вибором функцій та базовою перевіркою коректності даних
3	Складні цикли	Обчислення факторіалу, піднесення до степеню та знаходження сум числових рядів за допомогою циклічних конструкцій. Додавання повторного виконання обчислень	Реалізовано циклічні алгоритми математичних обчислень і повтор роботи програми
4	Рекурентні співвідношення	Обчислення чисел Фібоначчі, рекурентних послідовностей, наближеного обчислення функцій через рекурентні формули.	Додано підтримку рекурентних формул і послідовностей
5	Рекурсивні функції	Рекурсивна реалізація факторіалу, степеню, обчислення рядів Тейлора (sin, cos, exp). Порівняння з циклічними реалізаціями	Підсистема рекурсивних математичних функцій і тестування точності

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
6	Масиви	Збереження результатів обчислень у масивах. Обчислення статистики (середнє, мін, макс). Табулювання функцій	Таблиці значень функцій та аналіз результатів
7	Матриці	Реалізація операцій над матрицями (додавання, множення). Обчислення визначника матриці довільної вимірності. Табличні обчислення	Бібліотека базових матричних математичних операцій
8	Динамічна пам'ять і покажчики	Динамічні масиви для табулювання функцій. Динамічні матриці довільного розміру. Передача даних через покажчики	Гнучка бібліотека з підтримкою змінних розмірів даних
9	Обробка рядків	Парсинг математичних виразів (наприклад " $\sin(x)+x^2$ "). Вибір функцій через текстові команди. Формування текстових звітів	Текстовий інтерфейс для введення математичних виразів
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження результатів у файл. Завантаження параметрів обчислень. Експорт таблиць значень функцій. Об'єднання всіх модулів. Меню бібліотеки математичних функцій. Тести точності (порівняння з <code>std::math</code>). Документація	Підтримка імпорту/експорту результатів обчислень. Повноцінний консольний застосунок "Бібліотека математичних обчислень"

3.1.2 Картка проєкту «Програмна система статистичного аналізу даних»

Мета проєкту	Реалізувати базові розрахунки статистичних величин (середнє арифметичне, медіана, стандартне відхилення, мінімум, максимум) для заданого набору даних
Результат виконання	Консольний застосунок із функціями розрахунку статистичних величин, прикладами їх використання, тести точності, імпорт/експорт даних у файли.
Складність	Середня

Таблиця 3.2 – Інкременти проекту «Програмна система статистичного аналізу даних»

№	Тема лаб. роботи	Що додається до проекту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення набору числових даних з клавіатури. Обчислення суми та середнього арифметичного. Виведення результатів у консоль	Програма виконує базовий статистичний розрахунок середнього значення для введених даних
2	Розгалужені процеси	Вибір статистичної операції (середнє, мінімум, максимум). Перевірка коректності кількості даних. Обробка помилкового вводу	Інтерактивне меню вибору статистичних операцій з перевіркою даних
3	Складні цикли	Обчислення мінімального та максимального значення. Підрахунок кількості елементів. Повторний аналіз різних наборів даних	Реалізовано циклічну обробку наборів даних та базові статистичні показники
4	Рекурентні співвідношення	Покрокове обчислення середнього значення через рекурентну формулу. Обчислення накопичуваної суми та дисперсії	Додано обчислення статистичних величин через рекурентні алгоритми
5	Рекурсивні функції	Рекурсивний пошук мінімуму/максимуму. Рекурсивне обчислення суми елементів. Порівняння з ітераційними алгоритмами	Підсистема рекурсивного обчислення статистичних показників
6	Масиви	Збереження набору даних у масиві. Обчислення медіани (сортування). Обчислення стандартного відхилення	Аналіз статистичних показників для масиву даних
7	Матриці	Робота з двовимірними наборами даних. Обчислення статистики для рядків і стовпців. Табличне представлення даних	Підтримка статистичного аналізу табличних даних

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
8	Динамічна пам'ять і покажчики	Динамічне створення масивів довільного розміру. Передача масивів у функції через покажчики. Динамічні таблиці даних	Гнучка система аналізу статистичних даних змінного розміру
9	Обробка рядків	Введення даних у вигляді рядка (наприклад "1.2 3.4 5.6"). Перетворення рядка у числовий масив. Формування текстового звіту	Текстовий інтерфейс введення та форматування статистичних результатів
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Зчитування набору даних із файлу. Збереження результатів статистичного аналізу. Експорт таблиць у файл. Об'єднання всіх функцій статистичного аналізу. Додавання меню вибору методів. Тести точності та приклади використання	Імпорт/експорт статистичних даних і результатів аналізу. Повноцінна консольна система статистичного аналізу даних

3.1.3 Картка проєкту «Програмна бібліотека комбінаторних алгоритмів та прикладних задач»

Мета проєкту	Реалізувати набір алгоритмів для комбінаторних задач (перестановки, комбінації, розміщення)
Результат виконання	Консольний застосунок із функціями реалізації комбінаторних алгоритмів, прикладами їх використання, тести точності, імпорт/експорт даних у файли.
Складність	Середня

Таблиця 3.3 – Інкременти проєкту «Програмна бібліотека комбінаторних алгоритмів та прикладних задач»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення параметрів n та k. Обчислення факторіалу. Реалізація базових формул	Програма обчислює факторіал та прості комбінаторні значення

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
		комбінаторики. Виведення результатів	
2	Розгалужені процеси	Вибір типу комбінаторної задачі (перестановки, комбінації, розміщення). Перевірка коректності n і k . Обробка помилок введення	Інтерактивне меню вибору комбінаторних алгоритмів
3	Складні цикли	Ітераційне обчислення факторіалу. Обчислення кількості перестановок, комбінацій, розміщень через цикли. Повторні обчислення	Реалізовано циклічні алгоритми комбінаторних обчислень
4	Рекурентні співвідношення	Обчислення біноміальних коефіцієнтів через рекурентні формули. Реалізація трикутника Паскаля	Додано рекурентні комбінаторні співвідношення
5	Рекурсивні функції	Генерація перестановок рекурсією. Генерація комбінацій рекурсивним методом. Обчислення факторіалу рекурсивно	Підсистема рекурсивної генерації комбінаторних варіантів
6	Масиви	Збереження перестановок у масивах. Генерація всіх комбінацій набору. Підрахунок кількості варіантів	Робота з наборами комбінаторних результатів у масивах
7	Матриці	Представлення комбінацій у вигляді таблиць. Побудова таблиці біноміальних коефіцієнтів. Аналіз комбінаторних структур	Табличні комбінаторні обчислення
8	Динамічна пам'ять і покажчики	Динамічне створення масивів для великих наборів перестановок. Передача комбінаторних структур через покажчики	Гнучка бібліотека генерації комбінаторних наборів довільного розміру

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
9	Обробка рядків	Генерація перестановок символів рядка. Комбінаторні операції над текстовими даними. Формування текстового представлення результатів	Підтримка комбінаторних алгоритмів для рядкових даних
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження згенерованих комбінацій у файл. Завантаження параметрів задачі. Експорт результатів. Об'єднання алгоритмів перестановок, комбінацій і розміщень. Меню бібліотеки комбінаторних задач. Тести точності та приклади використання	Імпорт/експорт комбінаторних даних. Повноцінна програмна бібліотека комбінаторних алгоритмів

3.1.4 Картка проєкту «Програмна бібліотека елементарних алгоритмів обчислювальної геометрії та прикладних задач»

Мета проєкту	Реалізувати набір базових алгоритмів для роботи з геометричними об'єктами та задачами прикладної геометрії
Результат виконання	Консольний застосунок із функціями реалізації алгоритмів обчислювальної геометрії, прикладами їх використання, тести точності, імпорт/експорт даних у файли.
Складність	Середня

Таблиця 3.4 – Інкременти проєкту «Програмна бібліотека елементарних алгоритмів обчислювальної геометрії та прикладних задач»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення координат точок. Обчислення відстані між двома точками. Обчислення периметра та площі простих фігур	Програма виконує базові геометричні обчислення для точок і відрізків

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
2	Розгалужені процеси	Визначення типу трикутника за координатами. Перевірка належності точки відрізка. Вибір геометричної операції	Інтерактивний вибір геометричних задач з умовною логікою
3	Складні цикли	Обчислення довжини ламаної. Периметр багатокутника. Табулювання геометричних обчислень для набору точок	Обробка множини геометричних точок у циклах
4	Рекурентні співвідношення	Покрокове обчислення площі багатокутника через рекурентне додавання трикутників. Накопичувальні геометричні обчислення	Додано рекурентні алгоритми обчислення геометричних характеристик
5	Рекурсивні функції	Рекурсивне розбиття багатокутника на трикутники. Рекурсивний пошук мінімальної відстані між точками	Рекурсивні алгоритми обчислювальної геометрії
6	Масиви	Збереження координат точок у масивах. Обчислення площі багатокутника за масивом точок. Пошук найближчих точок	Робота з наборами геометричних об'єктів
7	Матриці	Використання матриць для афінних перетворень (зсув, масштабування). Табличне представлення координат	Реалізовано базові геометричні перетворення
8	Динамічна пам'ять і покажчики	Динамічні масиви точок. Обробка багатокутників довільної кількості вершин. Передача структур через покажчики	Гнучка бібліотека геометричних об'єктів змінного розміру
9	Обробка рядків	Введення точок у вигляді рядка. Парсинг координат. Формування текстового опису геометричних фігур	Текстовий інтерфейс для роботи з геометричними даними

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Зчитування координат із файлу. Збереження результатів обчислень. Експорт геометричних даних. Об'єднання алгоритмів роботи з точками, відрізками, багатокутниками. Загальне меню бібліотеки. Тести точності	Імпорт/експорт геометричних об'єктів .Повноцінна бібліотека алгоритмів обчислювальної геометрії

3.1.5 Картка проєкту «Програмна система керування книжковим фондом (бібліотека книг)»

Мета проєкту	Програмне моделювання роботи бібліотеки: каталог книг, каталог користувачів, облік видач і повернень книжок
Результат виконання	Консольний застосунок із функціями створення та ведення каталогів книг та користувачів, пошуком, обліком видач і повернень книжок, звітами популярності, імпорт/експорт даних у файли.
Складність	Середня

Таблиця 3.5 – Інкременти проєкту «Програмна система керування книжковим фондом (бібліотека книг)»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та операції введення/виведення	Введення інформації про книгу (назва, автор, рік). Виведення списку книг у консоль. Просте меню програми	Програма дозволяє створювати та переглядати базовий каталог книг
2	Розгалужені процеси	Меню вибору дій (додати книгу, переглянути, видалити). Перевірка коректності введених даних. Пошук книги за назвою	Інтерактивна система керування каталогом книг

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
3	Складні цикли	Перегляд списку книг. Пошук книги у каталозі. Реалізація повторного виконання операцій	Каталог книг з можливістю перегляду і пошуку
4	Рекурентні співвідношення	Облік кількості виданих книг. Накопичення статистики використання. Обчислення популярності книг	Додано базову статистику використання книжкового фонду
5	Рекурсивні функції	Рекурсивний пошук книги у каталозі. Рекурсивний перегляд списку користувачів	Реалізовано рекурсивні алгоритми обробки каталогів
6	Масиви	Масив книг і масив користувачів. Додавання та видалення записів. Облік видачі книг користувачам	Каталоги книг і користувачів на основі масивів
7	Матриці	Таблиця видач книг користувачам (книга × користувач). Облік повернень. Формування звітів популярності	Реалізовано облік видач і повернень книжок
8	Динамічна пам'ять і покажчики	Динамічні каталоги книг і користувачів. Розширення бібліотеки під час роботи. Передача структур через покажчики	Гнучка система керування бібліотекою довільного розміру
9	Обробка рядків	Пошук книг за частиною назви або автором. Форматування записів каталогу. Введення текстових даних користувачів	Розширений текстовий пошук у каталогах
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження каталогу книг і користувачів у файли. Завантаження даних при запуску. Збереження історії видач. Об'єднання каталогів книг, користувачів і обліку видач. Формування звітів популярності. Загальне меню системи	Постійне зберігання бібліотечного фонду. Повноцінна система керування книжковим фондом

3.1.6 Картка проекту «Програмна система обліку студентів та їх успішності»

Мета проекту	Реалізувати набір алгоритмів для ведення журналу успішності студентів з розрахунком їх рейтингів і відвідувань
Результат виконання	Консольний застосунок із матрицею «студент × предмет», рейтингами, звітами, пошуком, імпорт/експорт даних у файли.
Складність	Середня

Таблиця 3.6 – Інкременти проекту «Програмна система обліку студентів та їх успішності»

№	Тема лаб. роботи	Що додається до проекту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення інформації про студента (ПІБ, група). Введення оцінок з одного предмета. Виведення списку студентів	Програма формує базовий список студентів з оцінками
2	Розгалужені процеси	Вибір операцій (додати студента, додати оцінку, переглянути). Перевірка коректності оцінок. Визначення рівня успішності	Інтерактивне меню керування журналом успішності
3	Складні цикли	Обчислення середнього балу студента. Пошук мінімального та максимального балу. Перегляд списку студентів	Реалізовано базові розрахунки успішності студентів
4	Рекурентні співвідношення	Покрокове обчислення середнього рейтингу. Накопичення балів відвідувань. Обчислення загального рейтингу	Додано алгоритми накопичення рейтингу студентів
5	Рекурсивні функції	Рекурсивний пошук студента у списку. Рекурсивний перегляд журналу оцінок	Рекурсивна обробка журналу успішності
6	Масиви	Масив студентів. Масив оцінок для кожного студента. Обчислення середнього балу по групі	Журнал успішності на основі масивів

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
7	Матриці	Матриця «студент × предмет». Обчислення середнього балу по предмету. Формування таблиці успішності	Повноцінний журнал оцінок з кількох предметів
8	Динамічна пам'ять і покажчики	Динамічне створення списків студентів і предметів. Розширення журналу під час роботи	Гнучка система обліку студентів довільного розміру
9	Обробка рядків	Пошук студента за ПІБ. Формування текстових звітів. Обробка назв предметів	Розширений текстовий пошук і звітність
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження журналу успішності у файл. Завантаження даних. Експорт звітів рейтингів. Об'єднання обліку студентів, предметів, оцінок і відвідувань. Формування рейтингів. Загальне меню системи	Імпорт/експорт даних журналу успішності. Повноцінна система обліку студентів та їх успішності

3.1.7 Картка проєкту «Програмна система бронювання і купівлі квитків на події (концерти, фестивалі, вистави, екскурсії, конференції тощо)»

Мета проєкту	Реалізувати набір алгоритмів для ведення журналу успішності студентів з розрахунком їх рейтингів і відвідувань
Результат виконання	Консольний застосунок із матрицею «студент × предмет», рейтингами, звітами, пошуком, імпорт/експорт даних у файли.
Складність	Середня

Таблиця 3.7 – Інкременти проєкту «Програмна система бронювання і купівлі квитків на події (концерти, фестивалі, вистави, екскурсії, конференції тощо)»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення інформації про подію (назва, дата, ціна квитка, кількість місць). Виведення списку подій. Просте меню	Програма формує базовий каталог подій

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
2	Розгалужені процеси	Вибір дій (додати подію, переглянути, забронювати квиток). Перевірка наявності вільних місць. Обробка помилок введення	Інтерактивне меню системи бронювання
3	Складні цикли	Перегляд каталогу подій. Пошук події. Бронювання кількох квитків. Повторне виконання операцій	Реалізовано циклічну обробку каталогу та бронювання
4	Рекурентні співвідношення	Накопичення кількості проданих квитків. Покрокове обчислення виручки. Облік доступних місць	Додано обчислення статистики продажів
5	Рекурсивні функції	Рекурсивний пошук події у каталозі. Рекурсивний перегляд списку бронювань	Рекурсивна обробка каталогу подій
6	Масиви	Масив подій. Масив бронювань. Облік кількості вільних місць для кожної події	Каталог подій і бронювань на основі масивів
7	Матриці	Матриця «подія × місія» або «подія × користувач». Облік зайнятих місць. Формування таблиці продажів	Реалізовано систему бронювання місць
8	Динамічна пам'ять і покажчики	Динамічні списки подій і бронювань. Додавання нових подій під час роботи	Гнучка система бронювання довільного розміру
9	Обробка рядків	Пошук подій за назвою або датою. Введення даних користувача. Формування текстових квитків	Розширений текстовий інтерфейс бронювання
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження каталогу подій і бронювань. Завантаження даних. Збереження звітів виручки. Об'єднання каталогу подій, бронювання, обліку продажів. Підрахунок загальної виручки. Загальне меню системи	Імпорт/експорт даних системи продажу квитків. Повноцінна система бронювання і купівлі квитків

3.1.8 Картка проекту «Програмна система бронювання і купівлі авіаквитків»

Мета проекту	Імітувати бронювання і продаж квитків на транспортні рейси
Результат виконання	Консольний застосунок із функціями створення та ведення масивів рейсів, пасажирів, бронювання, пошуку за напрямками, підрахунку заповнюваності та доходу, збереженням у текстових і бінарних файлах.
Складність	Середня

Таблиця 3.8 – Інкременти проекту «Програмна система бронювання і купівлі авіаквитків»

№	Тема лаб. роботи	Що додається до проекту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення інформації про рейс (пункт відправлення, пункт прибуття, дата, ціна, кількість місць). Виведення списку рейсів. Просте меню	Програма дозволяє створювати та переглядати базовий каталог транспортних рейсів
2	Розгалужені процеси	Вибір дій (додати рейс, переглянути, знайти рейс, забронювати квиток). Перевірка наявності місць. Обробка помилкового вводу	Інтерактивна система керування рейсами та бронюванням
3	Складні цикли	Перегляд списку рейсів. Пошук рейсів за напрямком. Бронювання кількох місць. Повторне виконання операцій	Реалізовано циклічну обробку каталогу рейсів і бронювання квитків
4	Рекурентні співвідношення	Накопичення кількості проданих квитків. Покрокове обчислення заповнюваності рейсу. Підрахунок доходу	Додано статистику продажу квитків
5	Рекурсивні функції	Рекурсивний пошук рейсу у списку. Рекурсивний перегляд бронювань	Рекурсивна обробка списків рейсів і пасажирів
6	Масиви	Масив рейсів. Масив пасажирів. Облік заброньованих місць	Каталог рейсів та бронювань на основі масивів

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
7	Матриці	Матриця «рейс × місія». Облік зайнятих місць. Формування таблиці заповнюваності	Реалізовано систему бронювання місць
8	Динамічна пам'ять і показники	Динамічні списки рейсів і пасажирів. Додавання нових рейсів під час роботи	Гнучка система бронювання довільного розміру
9	Обробка рядків	Пошук рейсів за напрямком. Введення ПІБ пасажирів. Формування текстового квитка	Розширений текстовий пошук і формування квитків
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження рейсів і бронювань у файл. Завантаження даних. Експорт звітів доходу. Об'єднання каталогу рейсів, пасажирів, бронювання. Підрахунок заповнюваності та доходу	Імпорт/експорт даних системи бронювання. Повноцінна система бронювання транспортних квитків

3.1.9 Картка проєкту «Програмний менеджер особистих фінансів»

Мета проєкту	Облік доходів, витрат та планування особистого бюджету
Результат виконання	Консольний застосунок з категоріями транзакцій, звітами, балансом, прогнозом, імпорт/експорт даних у файли.
Складність	Середня

Таблиця 3.9 – Інкременти проєкту «Програмний менеджер особистих фінансів»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення доходів і витрат. Обчислення поточного балансу. Виведення списку операцій	Програма виконує базовий облік особистих фінансів
2	Розгалужені процеси	Вибір типу операції (дохід/витрата). Категорії витрат. Перевірка введених значень	Інтерактивний облік фінансових операцій

№	Тема лаб. роботи	Що додається до проекту	Результат інкременту
3	Складні цикли	Перегляд історії транзакцій. Підрахунок загальних доходів і витрат. Повторне виконання операцій	Реалізовано циклічну обробку фінансових даних
4	Рекурентні співвідношення	Накопичення балансу за періодами. Покрокове обчислення змін бюджету	Додано аналіз динаміки балансу
5	Рекурсивні функції	Рекурсивний пошук транзакцій. Рекурсивний підрахунок суми витрат	Рекурсивний аналіз фінансових операцій
6	Масиви	Масив транзакцій. Збереження категорій витрат. Підрахунок статистики	Журнал фінансових операцій
7	Матриці	Матриця «категорія × місяць». Аналіз витрат по категоріях	Таблична фінансова аналітика
8	Динамічна пам'ять і покажчики	Динамічний список транзакцій. Додавання нових категорій	Гнучка система обліку фінансів
9	Обробка рядків	Пошук операцій за категорією. Формування текстових звітів	Розширений пошук фінансових даних
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження фінансових даних. Завантаження історії операцій. Об'єднання транзакцій, категорій, аналітики. Формування звітів	Імпорт/експорт фінансових даних. Повноцінний менеджер особистих фінансів

3.1.10 Картка проекту «Програмна система для планування бюджету домогосподарства»

Мета проекту	Облік доходів, витрат та планування бюджету сім'ї
Результат виконання	Консольний застосунок для визначення доходів, обліку витрат, визначенням короткострокових і довгострокових фінансових цілей, розподілом бюджету, контролю фактичних витрат та аналізом відхилень від плану, імпортом/експортом даних у файли.
Складність	Середня

Таблиця 3.10 – Інкременти проекту «Програмна система для планування бюджету домогосподарства»

№	Тема лаб. роботи	Що додається до проекту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення доходів членів сім'ї. Введення витрат. Обчислення балансу	Базовий облік бюджету домогосподарства
2	Розгалужені процеси	Категорії витрат (їжа, транспорт, комунальні). Вибір операцій	Інтерактивне керування сімейним бюджетом
3	Складні цикли	Підрахунок витрат за категоріями. Перегляд списку витрат	Контроль витрат домогосподарства
4	Рекурентні співвідношення	Планування бюджету по місяцях. Накопичення заощаджень	Додано планування бюджету
5	Рекурсивні функції	Рекурсивний пошук витрат. Аналіз бюджету	Рекурсивний аналіз фінансових даних
6	Масиви	Масив доходів і витрат. Збереження категорій	Журнал сімейного бюджету
7	Матриці	Матриця «місяць × категорія». План/факт витрат	Таблиця бюджету
8	Динамічна пам'ять і показники	Динамічний список витрат і доходів	Гнучкий бюджет домогосподарства
9	Обробка рядків	Пошук витрат за категоріями. Формування звітів	Розширена аналітика бюджету
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження бюджету. Завантаження планів Об'єднання планування, аналізу, звітів	Імпорт/експорт бюджету. Повноцінна система бюджету домогосподарства

3.1.11 Картка проекту «Програмна система для симуляції банківських операцій»

Мета проекту	Моделювання роботи банку (створення й управління клієнтськими рахунками, виконання банківських операцій)
Результат виконання	Консольний застосунок для роботи з рахунками, операцій з коштами, конвертації валют за заданим курсом, аналітики, робота з даними (клієнти, рахунки, транзакції), імпорт/експорт у файли.
Складність	Середня

Таблиця 3.11 – Інкременти проекту «Програмна система для симуляції банківських операцій»

№	Тема лаб. роботи	Що додається до проекту	Результат інкременту
1	Базові обчислення та введення/виведення	Створення рахунку клієнта з ПІБ, номером рахунку та початковим балансом. Виведення інформації про рахунок	Програма формує базову базу клієнтських рахунків
2	Розгалужені процеси	Вибір операцій: депозит, зняття, перегляд балансу. Перевірка достатності коштів	Інтерактивне меню банківських операцій
3	Складні цикли	Перегляд історії операцій. Виконання декількох транзакцій підряд. Підрахунок загальної суми депозитів/зняттів	Реалізовано циклічну обробку рахунків
4	Рекурентні співвідношення	Накопичення відсотків по рахунках. Покроковий розрахунок зміни балансу	Додано алгоритми накопичення коштів і аналітики
5	Рекурсивні функції	Рекурсивний пошук клієнта та історії транзакцій	Рекурсивна обробка бази клієнтів
6	Масиви	Масив клієнтів. Масив операцій для кожного рахунку	База даних клієнтів і операцій
7	Матриці	Матриця «клієнт × операції». Формування історії транзакцій	Табличне відображення операцій

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
8	Динамічна пам'ять і показники	Динамічне додавання нових клієнтів і рахунків під час роботи	Гнучка система банківських операцій
9	Обробка рядків	Пошук клієнтів за ПІБ або номером рахунку. Формування звітів	Розширений текстовий пошук і звіти
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження даних клієнтів та транзакцій. Завантаження з файлу. Об'єднання обліку рахунків, транзакцій, відсотків та звітів	Імпорт/експорт даних банку. Повноцінна симуляція банківських операцій

3.1.12 Картка проєкту «Програмна система для обробки даних про кандидатів і вакансії (кадрове агентство)»

Мета проєкту	Імітація підбору персоналу за заданими параметрами та формування шорт-листів (скорочений пріоритетний список відібраних кандидатів)
Результат виконання	Консольний застосунок з масивом кандидатів/вакансій, матричним скорингом (метод оцінювання кандидатів), пошуком, фільтрами, звітами, експортом/імпортом у файли.
Складність	Середня

Таблиця 3.12 – Інкременти проєкту «Програмна система для обробки даних про кандидатів і вакансії (кадрове агентство)»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення інформації про кандидатів і вакансії (ПІБ, навички, посада, зарплатні очікування). Виведення списку кандидатів і вакансій	Програма формує базовий каталог кандидатів і вакансій

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
2	Розгалужені процеси	Вибір дій: додати кандидата/вакансію, переглянути, знайти відповідність. Перевірка коректності введення	Інтерактивне меню керування базою даних
3	Складні цикли	Перегляд списку кандидатів і вакансій. Пошук за навичками та зарплатою	Реалізовано циклічну обробку даних
4	Рекурентні співвідношення	Розрахунок скорингових оцінок кандидатів. Накопичення рейтингів	Додано алгоритми ранжування кандидатів
5	Рекурсивні функції	Рекурсивний пошук кандидата/вакансії за критеріями	Рекурсивна обробка даних для підбору
6	Масиви	Масив кандидатів. Масив вакансій. Збереження ключових показників	Каталог кандидатів і вакансій на основі масивів
7	Матриці	Матриця «кандидат × вакансія» з оцінками відповідності	Формування шорт-листів і таблиці скорингу
8	Динамічна пам'ять і покажчики	Динамічне додавання нових кандидатів/вакансій під час роботи	Гнучка база даних кадрового агентства
9	Обробка рядків	Пошук кандидатів за ПІБ, вакансій за посадою. Формування текстових звітів	Розширений пошук і звітність
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження бази кандидатів і вакансій. Завантаження та експорт. Об'єднання введення, пошуку, скорингу та звітів	Імпорт/експорт даних кадрового агентства. Повноцінна система підбору персоналу

3.1.13 Картка проекту «Програмна система для прогнозування попиту на продукцію»

Мета проекту	Розрахунок прогнозу попиту за історичними даними
Результат виконання	Консольний застосунок для збереження та обробки даних про попит, математичні розрахунки, порівняння фактичних і прогнозних значень, підрахунок середньої похибки прогнозу, фільтри, звіти, експорт/імпорт у файли.
Складність	Середня

Таблиця 3.13 – Інкременти проекту «Програмна система для прогнозування попиту на продукцію»

№	Тема лаб. роботи	Що додається до проекту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення історичних даних про попит на продукцію. Виведення списку даних	Програма формує базу історичних даних
2	Розгалужені процеси	Вибір типу продукції, періоду прогнозування. Перевірка введених даних	Інтерактивне меню прогнозування
3	Складні цикли	Підрахунок середнього попиту, пошук максимумів і мінімумів	Реалізовано обробку історичних даних
4	Рекурентні співвідношення	Використання ковзного середнього, накопичувальні обчислення прогнозу	Додано алгоритми прогнозування
5	Рекурсивні функції	Рекурсивний підрахунок прогнозу для наступних періодів	Рекурсивна обробка даних прогнозу
6	Масиви	Масив даних попиту. Масив прогнозних значень	Збереження історії і прогнозів
7	Матриці	Матриця «період × товар». Порівняння фактичного і прогнозного попиту	Аналітичні таблиці прогнозу
8	Динамічна пам'ять і покажчики	Динамічне додавання нових товарів і періодів	Гнучка система прогнозування
9	Обробка рядків	Пошук товарів за назвою, формування звітів	Розширений текстовий аналіз

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження даних і прогнозів у файли. Завантаження для аналізу. Об'єднання введення, обчислень, аналізу та звітів	Імпорт/експорт даних прогнозування. Повноцінна система прогнозування попиту

3.1.14 Картка проєкту «Програмна система аналітики ІТ ринку праці»

Мета проєкту	Збір та аналіз даних про вакансії, вимоги і зарплати в ІТ
Результат виконання	Консольний застосунок для збору і збереження даних про вакансії, фільтрації та пошуку вакансій за критеріями, аналізу навичок, що найчастіше зустрічаються, візуалізації аналітики ринку «технологія → кількість вакансій → середня зарплата», розрахунку рейтингів і динаміки попиту, формування звітів.
Складність	Середня

Таблиця 3.14 – Інкременти проєкту «Програмна система аналітики ІТ ринку праці»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення вакансій (назва посади, зарплата, технології). Виведення списку вакансій	Програма формує базу вакансій
2	Розгалужені процеси	Вибір дій: додати вакансію, переглянути, пошук за технологіями	Інтерактивне меню аналітики
3	Складні цикли	Підрахунок середньої зарплати по технології. Пошук вакансій за критеріями	Реалізовано статистичний аналіз вакансій
4	Рекурентні співвідношення	Розрахунок рейтингу технологій і трендів за періодами	Додано алгоритми аналітики ринку
5	Рекурсивні функції	Рекурсивний пошук вакансій за технологією та зарплатою	Рекурсивний аналіз бази вакансій
6	Масиви	Масив вакансій. Масив технологій. Збереження зарплат	Каталог вакансій і технологій

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
7	Матриці	Матриця «технологія × кількість вакансій». Матриця «технологія × середня зарплата»	Аналітичні таблиці ринку праці
8	Динамічна пам'ять і показники	Динамічне додавання нових вакансій і технологій	Гнучка аналітична система
9	Обробка рядків	Пошук вакансій за ключовими словами. Формування звітів	Розширена аналітика та пошук
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження і завантаження даних про вакансії. Об'єднання бази вакансій, аналітики, рейтингів і звітів	Імпорт/експорт аналітичних даних. Повноцінна система аналітики ІТ ринку

3.1.15 Картка проєкту «Персональний електронний органайзер студента»

Мета проєкту	Планування навчальних справ, дедлайнів і нотаток
Результат виконання	Консольний застосунок з календарем, нагадуваннями, пошуком подій, збереженням у текстових і бінарних файлах.
Складність	Середня

Таблиця 3.15 – Інкременти проєкту «Персональний електронний органайзер студента»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення завдань, подій, дедлайнів. Виведення списку подій	Програма дозволяє створювати базовий календар студента
2	Розгалужені процеси	Вибір типу дії: додати подію, переглянути, видалити. Перевірка правильності вводу	Інтерактивне меню органайзера
3	Складні цикли	Перегляд подій за датою, категорією. Повторне виконання операцій	Реалізовано циклічне управління подіями

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
4	Рекурентні співвідношення	Накопичення кількості завдань на день/тиждень. Розрахунок вільного часу	Додано базову аналітику завантаженості
5	Рекурсивні функції	Рекурсивний пошук подій за ключовими словами або датою	Рекурсивний пошук завдань
6	Масиви	Масив подій. Збереження категорій подій	Органайзер на основі масивів
7	Матриці	Матриця «день × година». Відображення завантаженості по годинах	Візуалізація календаря у табличному вигляді
8	Динамічна пам'ять і покажчики	Динамічне додавання подій під час роботи	Гнучкий електронний органайзер
9	Обробка рядків	Пошук подій за ключовими словами. Формування звітів	Розширений пошук та звіти
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження і завантаження подій із файлів. Об'єднання введення, пошуку, календаря, звітів	Імпорт/експорт даних органайзера. Повноцінний електронний органайзер студента

3.1.16 Картка проєкту «Програмна система «Журнал куратора академічної групи»»

Мета проєкту	Ведення особових справ студента з обліком та аналізом відвідуваності, успішності і дисциплінарних записів групи
Результат виконання	Консольний застосунок з матрицею «студент × предмет», рейтингами, нотатками куратора, пошуком, імпортом/експортом у файли.
Складність	Середня

Таблиця 3.16 – Інкременти проєкту «Програмна система «Журнал куратора академічної групи»»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення студентів, предметів, оцінок,	Програма формує базовий журнал групи

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
		відвідуваності. Виведення матриці «студент × предмет»	
2	Розгалужені процеси	Вибір дії: додати/редагувати студента, оцінку, відвідуваність	Інтерактивне меню ведення журналу
3	Складні цикли	Підрахунок середніх оцінок. Аналіз відвідуваності. Перегляд рейтингу студентів	Реалізовано аналітику успішності
4	Рекурентні співвідношення	Накопичення сум оцінок для розрахунку рейтингу. Визначення середнього балу групи	Додано динамічні розрахунки рейтингів
5	Рекурсивні функції	Рекурсивний пошук студента за ПІБ або номером	Рекурсивний пошук у журналі
6	Масиви	Масив студентів. Масив предметів. Збереження оцінок та відвідуваності	Журнал на основі масивів
7	Матриці	Матриця «студент × предмет» з оцінками і відвідуваністю	Табличне відображення успішності
8	Динамічна пам'ять і покажчики	Динамічне додавання студентів і предметів	Гнучка система ведення журналу
9	Обробка рядків	Пошук студентів за ПІБ, предметів за назвою. Формування текстових звітів	Розширений пошук та аналітика
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження та завантаження журналу. Об'єднання всіх функцій: оцінки, відвідуваність, рейтинг, звіти	Імпорт/експорт даних журналу. Повноцінний журнал куратора академічної групи

3.1.17 Картка проекту «Програмна система моніторингу вебстатистики»

Мета проекту	Збір, зберігання та аналіз даних про відвідуваність вебресурсу (кількість відвідувачів, кількість переглядів сторінок, середній час перебування, джерела трафіку тощо), виявляти тренди та формувати звіти маркетингових стратегій
Результат виконання	Консольний застосунок для збору, збереження та визначення статистичних показників, аналіз джерел трафіку, аналітика та візуалізація у звітах, розрахунок рейтингів сайтів, фільтрація та пошук, звіти та імпорт/експорт у файли.
Складність	Середня

Таблиця 3.17 – Інкременти проекту «Програмна система моніторингу вебстатистики»

№	Тема лаб. роботи	Що додається до проекту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення даних про відвідуваність: користувачі, перегляди, час перебування	Програма формує базову статистику вебресурсу
2	Розгалужені процеси	Вибір дій: додати дані, переглянути статистику, аналізувати	Інтерактивне меню вебмоніторингу
3	Складні цикли	Підрахунок середніх показників, аналіз тенденцій. Повторна обробка даних	Реалізовано циклічний збір і обробку даних
4	Рекурентні співвідношення	Накопичення статистики за періодами. Розрахунок змін трафіку	Додано аналіз трендів відвідуваності
5	Рекурсивні функції	Рекурсивний пошук даних за джерелами трафіку чи датами	Рекурсивний аналіз статистики
6	Масиви	Масив відвідувань, переглядів, джерел трафіку	Збереження статистичних даних
7	Матриці	Матриця «день × показник» для аналізу трафіку	Таблична аналітика вебстатистики
8	Динамічна пам'ять і покажчики	Динамічне додавання нових показників та джерел	Гнучка система збору статистики

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
9	Обробка рядків	Пошук даних за джерелами, формування звітів	Розширений пошук і аналітика
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження даних у файл. Завантаження історії. Об'єднання всіх функцій: збір, обробка, аналітика, звіти	Імпорт/експорт вебстатистики. Повноцінна система моніторингу вебстатистики

3.1.18 Картка проєкту «Програмна система «Калькулятор подорожей»»

Мета проєкту	Розрахунок вартості подорожі з урахуванням транспорту, проживання, харчування, розваг та додаткових витрат, а також порівняти кілька варіантів маршруту чи бюджету
Результат виконання	Консольний застосунок для введення та зберігання параметрів подорожі, розрахунку вартості подорожі, порівняння декількох варіантів подорожі за ціною та тривалістю, імпорт/експорт у файли, звіт.
Складність	Середня

Таблиця 3.18 – Інкременти проєкту «Програмна система «Калькулятор подорожей»»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення параметрів подорожі: транспорт, проживання, харчування	Програма формує базову калькуляцію подорожі
2	Розгалужені процеси	Вибір дій: розрахунок вартості, порівняння варіантів. Перевірка введених даних	Інтерактивне меню калькулятора
3	Складні цикли	Обчислення вартості кожного етапу подорожі. Порівняння декількох маршрутів	Реалізовано циклічні обчислення вартості
4	Рекурентні співвідношення	Накопичення витрат по днях або категоріях	Додано аналіз бюджету подорожі

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
5	Рекурсивні функції	Рекурсивний розрахунок витрат на маршрути з пересадками	Рекурсивний аналіз маршрутів
6	Масиви	Масив варіантів маршрутів. Масив категорій витрат	Збереження даних про подорожі
7	Матриці	Матриця «день × категорія витрат». Порівняння маршрутів	Табличне представлення вартості подорожей
8	Динамічна пам'ять і покажчики	Динамічне додавання варіантів маршрутів під час роботи	Гнучка система калькулятора подорожей
9	Обробка рядків	Пошук маршрутів за ключовими словами, формування звітів	Розширений пошук і аналіз маршрутів
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження даних подорожей у файл. Завантаження історії. Об'єднання введення, обчислень, порівнянь та звітів	Імпорт/експорт калькуляцій подорожей. Повноцінний калькулятор подорожей

3.1.19 Картка проєкту «Програмна система для виявлення вразливих паролів»

Мета проєкту	Аналізує паролі користувачів на надійність та виявляє ті, що можуть бути легко зламані, використовуючи словникові атаки та базові алгоритми перевірки стійкості
Результат виконання	Консольний застосунок для читання паролів з файлів, перевірка паролів на відповідність різним критеріям, виявлення відповідності найпоширенішим паролям, обчислення оцінки надійності пароля, звіт, запис результатів перевірки у файл.
Складність	Середня

Таблиця 3.19 – Інкременти проекту «Програмна система для виявлення вразливих паролів»

№	Тема лаб. роботи	Що додається до проекту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення паролів вручну або з файлу. Виведення результатів перевірки	Програма формує базовий список паролів і оцінок
2	Розгалужені процеси	Перевірка пароля на довжину, наявність цифр, літер, символів	Інтерактивна перевірка на базові критерії безпеки
3	Складні цикли	Перевірка списку паролів. Підрахунок кількості слабких/сильних	Реалізовано циклічну обробку паролів
4	Рекурентні співвідношення	Порівняння пароля зі словником поширених паролів	Додано аналіз стійкості щодо словникових атак
5	Рекурсивні функції	Рекурсивний пошук паролів у словнику	Рекурсивна перевірка відповідності
6	Масиви	Масив паролів. Масив оцінок безпеки	Збереження даних для аналізу
7	Матриці	Матриця «пароль × критерій безпеки»	Табличне представлення оцінок
8	Динамічна пам'ять і покажчики	Динамічне додавання паролів і словників	Гнучка система перевірки паролів
9	Обробка рядків	Пошук слабких слів, формування текстових звітів	Розширена аналітика надійності паролів
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження результатів перевірки у файл. Завантаження словників. Об'єднання перевірки, аналізу, звітів	Імпорт/експорт даних перевірки паролів. Повноцінна система для оцінки надійності паролів

3.1.20 Картка проекту «Програмна система обліку гуманітарної допомоги під час війни»

Мета проекту	Планування та фактичний облік вантажів, відправників і отримувачів, формування звітів, виявлення невідповідності плану та факту, імпорту/експорту у файли
Результат виконання	Консольний застосунок для введення, редагування та видалення записів про вантажі, пошук і фільтрація даних, сортування та підрахунок статистики допомоги
Складність	Середня

Таблиця 3.20 – Інкременти проекту «Програмна система обліку гуманітарної допомоги під час війни»

№	Тема лаб. роботи	Що додається до проекту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення інформації про вантаж (назва, кількість, відправник, отримувач, дата). Виведення списку гуманітарної допомоги	Програма формує базовий реєстр гуманітарних вантажів
2	Розгалужені процеси	Вибір дій: додати вантаж, переглянути, редагувати, видалити. Перевірка коректності введених даних	Інтерактивна система керування обліком допомоги
3	Складні цикли	Перегляд списку вантажів. Пошук за отримувачем або відправником. Підрахунок кількості допомоги	Реалізовано циклічну обробку даних гуманітарної допомоги
4	Рекурентні співвідношення	Накопичення обсягів допомоги по регіонах або отримувачах. Підрахунок загальної кількості вантажів	Додано статистичний облік гуманітарної допомоги
5	Рекурсивні функції	Рекурсивний пошук записів за назвою вантажу або отримувачем	Рекурсивний аналіз реєстру допомоги
6	Масиви	Масив вантажів. Масив отримувачів. Збереження даних про кількість і типи допомоги	Реєстр гуманітарної допомоги на основі масивів

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
7	Матриці	Матриця «відправник × отримувач» або «регіон × тип допомоги». Аналіз розподілу	Таблична аналітика гуманітарної допомоги
8	Динамічна пам'ять і показники	Динамічне додавання нових записів про допомогу. Розширення бази даних	Гнучка система обліку гуманітарних вантажів
9	Обробка рядків	Пошук вантажів за назвою. Формування текстових звітів. Фільтрація за категоріями	Розширений пошук і звітність
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження реєстру гуманітарної допомоги. Завантаження даних. Експорт статистики. Об'єднання введення, пошуку, обліку, статистики та звітів	Імпорт/експорт даних гуманітарної допомоги. Повноцінна система обліку гуманітарної допомоги

3.1.21 Картка проєкту «Програмна система моделювання тактичних операцій з БпЛА»

Мета проєкту	Симулятор планування та виконання місій БпЛА: опис парку дронів, маршрутизація через точки, оцінка енерговитрат і ризиків, планування черги вильотів та підготовка звітів
Результат виконання	Консольний застосунок для створення бази парку БпЛА, місії, карта-грід, no-fly зони, планування маршрутів (DFS + матричні витрати), стратегічні компаратори/ планувальники, парсинг текстових наказів, лог телеметрії, збереження/ завантаження у текстовому та бінарному форматах, звіти з тривалості, енергії та успішності роботи дронів
Складність	Середня

Таблиця 3.21 – Інкременти проекту «Програмна система моделювання тактичних операцій з БпЛА»

№	Тема лаб. роботи	Що додається до проекту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення інформації про БпЛА (назва, дальність, швидкість, запас енергії). Введення точок маршруту	Програма формує базу безпілотників і маршрутних точок
2	Розгалужені процеси	Вибір дій: створити місію, додати точку маршруту, переглянути місії. Перевірка можливості виконання маршруту	Інтерактивна система планування місій БпЛА
3	Складні цикли	Обхід маршруту через точки. Обчислення загальної дистанції. Розрахунок тривалості місії	Реалізовано циклічне планування маршруту
4	Рекурентні співвідношення	Покрокове накопичення витрат енергії. Обчислення залишку ресурсу БпЛА	Додано аналіз енерговитрат місії
5	Рекурсивні функції	Рекурсивний пошук маршруту між точками (DFS). Рекурсивна побудова місії	Рекурсивні алгоритми маршрутизації
6	Масиви	Масив БпЛА. Масив маршрутних точок. Масив місій	База даних дронів і місій
7	Матриці	Матриця вартості переходів між точками. Матриця ризиків або енерговитрат	Табличне планування маршрутів
8	Динамічна пам'ять і покажчики	Динамічне додавання нових місій, дронів і точок	Гнучка система моделювання місій
9	Обробка рядків	Парсинг текстових команд місії. Пошук БпЛА за назвою. Формування звітів	Текстове управління симулятором
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли. Фінальна інтеграція	Збереження місій, парку БпЛА, маршрутів. Завантаження сценаріїв. Об'єднання бази БпЛА, місій, маршрутів, аналітики та звітів	Імпорт/експорт даних симулятора. Повноцінна система моделювання тактичних операцій БпЛА

РОЗДІЛ 4 ІНКРЕМЕНТИ ПРОЄКТУ (ЛАБОРАТОРНІ РОБОТИ)

У даному розділі для кожного інкременту (лабораторної роботи) подано теоретичні відомості, приклад виконання типового завдання та контрольні запитання. Для прикладів обрано проєкт:

Назва проєкту: Програмна система конвертації одиниць вимірювання

Мета проєкту: реалізувати набір алгоритмів для конвертації одиниць вимірювання (довжина, маса, температура, час, швидкість тощо).

Результат виконання проєкту: консольний застосунок із функціями конвертації одиниць вимірювання, вибором категорій, прикладами використання, тестами точності, імпортом/експортом даних у файли.

Таблиця 4.1 – Інкременти проєкту «Програмна система конвертації одиниць вимірювання»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Типи даних. Функції введення - виведення. Обчислення виразів	Введення значення та вибір одиниць вимірювання. Реалізація простих конвертацій (метри–кілометри, грами–кілограми). Виведення результату	Програма виконує базову конвертацію одиниць вимірювання
2	Розгалужені обчислювальні процеси	Вибір категорії конвертації (довжина, маса, температура, час). Перевірка коректності введених даних	Інтерактивне меню вибору типу конвертації
3	Циклічні обчислювальні процеси	Повторне виконання конвертацій. Табулювання значень (наприклад 1–10 метрів у сантиметри)	Реалізовано циклічну обробку конвертацій
4	Рекурентні співвідношення	Покрокова конвертація через проміжні одиниці (мм→см→м→км). Накопичувальні перетворення	Додано багатокрокову конвертацію одиниць
5	Рекурсивні функції	Рекурсивна конвертація через ланцюжок одиниць. Рекурсивний пошук потрібного перетворення	Рекурсивна система перетворення одиниць
6	Одновимірні масиви	Масив підтримуваних одиниць. Масив коефіцієнтів	Каталог одиниць вимірювання

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
		конвертації. Обчислення кількох значень	
7	Багатовимірні масиви	Матриця коефіцієнтів конвертації між одиницями. Табличне перетворення значень	Таблична система конвертації
8	Динамічна пам'ять і покажчики	Динамічне додавання нових одиниць вимірювання. Розширення системи конвертації	Гнучка система підтримки одиниць
9	Обробка рядків	Введення одиниць текстом ("km", "kg"). Пошук відповідних одиниць. Формування звітів	Текстовий інтерфейс конвертації
10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання структур і масивів структур
11	Текстові та бінарні файли	Збереження списку одиниць і коефіцієнтів. Завантаження конфігурації	Імпорт/експорт даних конвертера

Таблиця 4.2 – Карта рефакторингу інкрементів

Інкремент	Що додано	Навчальне спрощення	Обмеження спрощення	Подальший рефакторинг
1	Типи даних, введення/виведення, арифметичні вирази, базові конвертації	Уся логіка розміщена у main(); розгалуження подано як готовий диспетчер	Велика відповідальність main(), обмежена розширюваність	В інкременті 2 меню, введення та конвертації винесено в окремі функції
2	Категорії конвертацій, if/switch, перевірка введення, функціональна декомпозиція	Меню виконується один раз; допоміжні функції введення використовують посилання як готовий засіб	Немає повторного виконання; посилання ще не є основною темою	В інкременті 3 додано цикли повторення і табулювання

3	do-while, for, таблиці конвертації, перевірка меж і кроку	Значення таблиці обчислюються незалежно за формулою $start + index * step$	Не демонструється залежність наступного члена від попереднього	В інкременті 4 таблицю перебудовано на рекурентне співвідношення
4	Рекурентне формування аргументу та накопиченої суми	Рекурентний процес реалізовано циклом	Це ще не рекурсивний виклик функції	В інкременті 5 додано рекурсивні функції та явний базовий випадок
5	Рекурсивна конвертація і рекурсивний друк ланцюжка	Послідовність мм $\rightarrow$ см $\rightarrow$ м $\rightarrow$ км зафіксована в коді	Додавання одиниці потребує зміни кількох функцій	В інкременті 6 назви та коефіцієнти перенесено в одновимірні масиви
6	Масиви одиниць і значень, пошук, сортування, бінарний пошук, статистика	Статичні масиви мають фіксований максимальний розмір; назви є <code>const char*</code>	Не можна додавати одиниці під час виконання; дані розміщені в окремих масивах	В інкременті 7 додано матрицю, в інкременті 8 — динамічні масиви
7	Двовимірна матриця коефіцієнтів і вкладені цикли	Розмір матриці визначено під час компіляції	Неможливе динамічне розширення	В інкременті 8 матрицю замінено на <code>double**</code> і додано розширення
8	Динамічна пам'ять, покажчики, посилання на покажчики, масив покажчиків, покажчики на функції	Ручне керування <code>new[]/delete[]</code> ; назви зберігаються як С-рядки	Ризик витоків і помилок копіювання, складний код роботи з текстом	В інкременті 9 С-рядки замінено на <code>std::string</code> ; в інкременті 10 дані об'єднано у структури
9	<code>std::string</code> , <code>getline()</code> , пошук за назвою, текстовий звіт	Назви й коефіцієнти зберігаються у паралельних масивах	Можлива розсинхронізація пов'язаних даних	В інкременті 10 введено структуру <code>Unit</code> і масив структур
10	Структура <code>Unit</code> , масив структур,	Масив структур і далі керується вручну; <code>union</code> не	Дані існують лише під час роботи програми	В інкременті 11 додано текстове

	передавання через const-посилання	використано без реальної потреби		й бінарне збереження
11	Текстові та бінарні файли, безпечне завантаження, перевірка потоків	Формат файлу не має версії схеми та контрольної суми	Обмежена сумісність майбутніх версій формату	Поза межами курсу: std::vector, RAII, розумні покажчики, модулі, версіонований формат

Як уніфіковано програмний код:

1. **Єдина структура файлів.** Кожен файл починається з блоку: назва проєкту, номер і тема інкременту, перелік доданої функціональності, навчальне спрощення та запланований рефакторинг.
2. **Єдиний стиль C++.** Використано відступ у чотири пробіли, фігурні дужки з нового рядка, camelCase для функцій і змінних, UPPER_SNAKE_CASE для констант. Вилучено using namespace std.
3. **Коротка функція main().** Починаючи з інкременту 2, main() лише налаштовує локаль і передає керування runApplication().
4. **SRP.** Введення, меню, перевірка, обчислення, пошук, сортування, виведення, керування пам'яттю та файлові операції розділено між окремими функціями.
5. **DRY.** Повторювані перевірки введення винесено в readInt(), readDouble(), readChoice(); коефіцієнти конвертації мають єдине джерело; матриця формується автоматично, а не дублюється вручну.
6. **Низька зв'язаність.** Відсутні глобальні змінні зі змінним станом. Дані передаються параметрами; функції обчислення не залежать від меню або консольного введення.
7. **Змістовні ідентифікатори.** Застосовано назви sourceIndex, targetIndex, factorsToMillimeters, convertedValues, cumulativeSum, loadedUnits тощо.
8. **Відсутність магічних чисел.** Межі, коефіцієнти, максимальні розміри, похибка порівняння та кількість початкових одиниць оформлено як іменовані константи.
9. **Перевірка даних.** Перевіряються тип введення, діапазони меню, межі масивів, додатність кроку й коефіцієнтів, дублікати назв, результати файлових операцій та допустимість кількості записів.
10. **Безпечна робота з файлами.** Завантаження виконується в тимчасовий масив; поточні дані замінюються лише після повного успішного читання. Бінарні рядки записуються як довжина та послідовність символів, а не як внутрішнє представлення std::string.

4.1 Лабораторна робота №1. Типи даних. Функції введення-виведення. Обчислення виразів

4.1.1 Мета та завдання

Мета роботи:

1. Ознайомитися з основними типами даних у мові C++ та їх властивостями.
2. Вивчити принципи введення та виведення даних за допомогою стандартних потоків.
3. Набути практичних навичок обчислення арифметичних виразів.
4. Засвоїти правила запису математичних виразів мовою програмування C++.

Завдання роботи:

1. Вивчити основні вбудовані типи даних C++ (`int`, `float`, `double`, `char` тощо).
2. Ознайомитися з використанням об'єктів стандартних потоків `cin` та `cout`, а також операторів видобування (`>>`) та вставки (`<<`).
3. Розробити програму, що вводять з клавіатури необхідні дані.
4. Реалізувати обчислення заданого арифметичного виразу.
5. Вивести результат обчислень на екран у зрозумілому форматі.
6. Перевірити правильність роботи програми на різних вхідних даних.

4.1.2 Теоретичні відомості

Структура програми

Програма мовою C++ складається з директив препроцесора, оголошень, функцій та операторів.

Для використання засобів стандартного введення і виведення підключають заголовний файл `iostream`:

```
#include <iostream>
```

Директива `#include` забезпечує підключення до програми оголошень стандартних засобів, зокрема об'єктів `std::cin` і `std::cout`.

Виконання програми починається з функції `main()`, яка є точкою входу:

```
int main()
{
    // оператори програми
    return 0;
}
```

Функція `main()` повертає ціле значення. Значення 0 зазвичай означає успішне завершення програми.

Тіло функції обмежується фігурними дужками `{ }`. Кожен оператор завершується символом крапки з комою `;`.

Змінні та типи даних

Змінна – це іменована ділянка оперативної пам'яті, призначена для зберігання даних. Кожна змінна має ім'я, тип і значення.

Змінну потрібно оголосити до її першого використання:

```
double value;  
double result;  
int choice;
```

Тип даних визначає:

- які значення може зберігати змінна;
- скільки пам'яті для неї виділяється;
- які операції можна виконувати над її значенням.

У таблиці 4.3 представлені типи даних мові програмування C/C++.

Таблиця 4.3 – Характеристика типів змінних

Тип	Назва типу	Обсяг оперативної пам'яті	Діапазон значень
char	Символьний (Цілий) тип	8 бітів	-128 ... +127
unsigned char	Беззнаковий цілий	8 бітів	0 ... 255
short int	Цілий короткий	16 бітів	-32768 ... +32767
unsigned short	Беззнаковий цілий короткий	16 бітів	0 ... 65535
int	Цілий	32 бітів	-2147483648 ... +2147483647
unsigned int	Беззнаковий цілий	32 бітів	0 ... 4 294 967 295
long (long int)	Цілий довгий	32 або 64 бітів	залежить від платформи
unsigned long	Цілий довгий беззнаковий	32 або 64 бітів	0 ... +429496729 або більше
long long	Цілий наддовгий	64 бітів	$\pm 9 \times 10^{18}$
float	З плаваючою крапкою	32 бітів	$3.4 \cdot 10^{-38} \dots 3.4 \cdot 10^{38}$
double	З плаваючою крапкою довгий	64 бітів	$1.7 \cdot 10^{-308} \dots 1.7 \cdot 10^{308}$
long double	З плаваючою крапкою довгий	80/96/128 бітів	$3.4 \cdot 10^{-4932} \dots 3.4 \cdot 10^{4932}$

Для виконання першого інкременту достатньо використовувати такі типи: int, double, char, bool.

Приклад оголошення та ініціалізації змінних:

```
double value = 0.0;
```

```
double result = 0.0;
int choice = 0;
```

Ініціалізація – це надання змінній початкового значення під час її оголошення. Змінні доцільно ініціалізувати, щоб уникнути використання невизначених значень.

Ідентифікатор – це ім'я змінної, функції або іншого програмного об'єкта.

Ідентифікатор:

- може містити латинські літери, цифри та символ підкреслення;
- не може починатися з цифри;
- не може збігатися із зарезервованим словом мови C++;
- має відображати призначення програмного об'єкта.

Доцільно використовувати зрозумілі імена:

value – початкове значення;
result – результат обчислення;
choice – номер обраної операції.

Імена на зразок a, b або x допустимі для коротких математичних прикладів, але в прикладній програмі краще використовувати змістовні назви.

Константа – це значення, яке не змінюється під час виконання програми.

Іменовані константи оголошують за допомогою ключового слова `const`:

```
const double METERS_IN_KILOMETER = 1000.0;
```

Використання іменованих констант підвищує зрозумілість коду та дає змогу уникати «магічних чисел» – числових значень, зміст яких не пояснено.

Наприклад, замість:

```
result = value / 1000.0;
```

можна записати:

```
const double METERS_IN_KILOMETER = 1000.0;
result = value / METERS_IN_KILOMETER;
```

Коментарі

Коментарі використовують для пояснення призначення окремих фрагментів програми. Коментар не виконується компілятором і не впливає на результат роботи програми.

Однорядковий коментар починається символами `//`:

```
double value; // значення, яке потрібно конвертувати
```

Багаторядковий коментар записується між символами `/*` і `*/`:

```
/*
```

Програма виконує конвертацію
одиниць вимірювання.
*/

Коментарі повинні пояснювати призначення даних або логіку обчислень, а не дослівно повторювати програмний код.

Арифметичні операції

Операція присвоєння записується символом =:

```
result = value / 1000.0;
```

Спочатку обчислюється вираз у правій частині оператора, після чого отримане значення записується у змінну, зазначену в лівій частині.

Операцію присвоєння не слід плутати з математичною рівністю. Після виконання нового присвоєння попереднє значення змінної замінюється новим.

Для обчислення виразів у C++ використовують арифметичні операції:

- + додавання;
- віднімання;
- * множення;
- / ділення;
- % остача від ділення цілих чисел.

Приклади:

```
double kilometers = meters / 1000.0;  
double meters = kilometers * 1000.0;  
double kilograms = grams / 1000.0;  
double grams = kilograms * 1000.0;
```

Під час ділення цілих чисел дробова частина результату відкидається:

```
int result = 5 / 2; // результат дорівнює 2
```

Щоб отримати дійсний результат, хоча б один операнд має бути дійсного типу:

```
double result = 5.0 / 2.0; // результат дорівнює 2.5
```

Саме тому в коефіцієнтах конвертації доцільно використовувати значення 1000.0, а не 1000.

Арифметичні вирази обчислюються відповідно до пріоритету операцій:

1. Вирази в круглих дужках;
2. Множення, ділення та остача від ділення;

3. Додавання і віднімання.

Наприклад:

```
double result = (value * 9.0 / 5.0) + 32.0;
```

Круглі дужки допомагають явно визначити порядок обчислень і покращують читабельність виразу.

Функції введення та виведення даних, математичні функції

Для виконання дій над даними користувача необхідно їх ввести з клавіатури. Для цього використовується стандартний потік вводу `stdin`. Для відображення результатів на екрані необхідно застосовувати стандартний потік виводу `stdout`. Введення та виведення даних здійснюється за допомогою функцій введення та виведення, формати яких приведені у таблиці 4.4. В таблиці не приведені функції консольного введення-виведення мови C, оскільки вони не є кросплатформні.

Таблиця 4.4 – Функції введення та виведення

Функція	Формат введення/виведення	Призначення	Приклад	Заголовний файл	Результат, що повертається функцією
<code>getchar()</code>	<code>int getchar(void);</code>	Введення символу із стандартного вхідного файлу <code>stdin</code> .	<code>int ch; ch=getchar();</code>	<code>stdio.h</code> або <code>cstdio</code>	Повертається код введеного символу або значення EOF
<code>putchar()</code>	<code>int putchar(int c);</code>	Повертається символ або значення EOF при невдалому виводу	<code>putchar(_getch());</code>	<code>stdio.h</code> або <code>cstdio</code>	Повертається символ або значення EOF при невдалому виводу
<code>gets_s()</code>	<code>char *gets_s(char *s);</code>	Введення із стандартного вхідного файлу <code>stdin</code> рядка і розміщення його по покажчику <code>*s</code>	<code>char line[80]; gets(line);</code>	<code>stdio.h</code> або <code>cstdio</code>	Покажчик на перший символ рядка або NULL при невдалому введенні
<code>puts()</code>	<code>int puts(const char *s);</code>	Виведення у стандартний вихідний файл <code>stdout</code> рядка і доповнення його символом нового рядка <code>'\n'</code>	<code>char b[80]; puts(b);</code>	<code>stdio.h</code> або <code>cstdio</code>	Повертається невід'ємне значення або EOF при невдалому виведенні

<code>printf()</code>	<code>int printf (const char *format, argument, ...);</code>	Виведення у стандартний вихідний файл <code>stdout</code> значень аргументів згідно з форматом виведення	<code>int a; float b; char str[10]; printf("%d %f %s", a, b, str);</code>	<code>stdio.h</code> або <code>cstdio</code>	Повертається кількість виведених байтів або від'ємне значення при невдалому виведенні
<code>scanf_s()</code>	<code>int scanf_s (const char *format, address, ...);</code>	Введення із стандартного вхідного файлу значень змінних згідно з форматом і розміщення їх за вказаними адресами у оперативній пам'яті.	<code>char AME[20]; int a; scanf("%s", AME); scanf("%d", &a);</code>	<code>stdio.h</code> або <code>cstdio</code>	Повертається кількість змінних, що отримали значення. При невдалому скануванні повертається значення EOF
<code>sprintf()</code>	<code>int sprintf (char *str, const char *format, argument, ...);</code>	Розміщення у рядку символів значень аргументів із пам'яті згідно з шаблоном. Використовується для перетворення типів даних	<code>char str[3]; sprintf(str, "%d", 13);</code>	<code>stdio.h</code> або <code>cstdio</code>	Повертається кількість виведених символів.
<code>sscanf_s()</code>	<code>int sscanf (рядок, фор-мат, arg1, ...);</code>	Читання із рядка у адреси пам'яті згідно з форматом шаблонів. Використовується для перетворення типів	<code>int a; float b; char str[5]; sscanf(str, "%d%", &a, &b);</code>	<code>stdio.h</code> або <code>cstdio</code>	Повертається кількість введених символів або EOF при невдалому введенні

За допомогою об'єктів класів потоків здійснюють введення та виведення, використовуючи операції лівого та правого зсуву згідно з табл. 4.5.

Таблиця 4.5 – Використання класів потоків для операцій вводу, виводу

Формат	Призначення	Приклад	Заголовний файл
<code>cin >></code>	Зумовлений потік введення, <code>>></code> - операція правого зсуву	<code>char name[20]; cin >> name;</code>	<code>istream</code> <code>iostream</code>
<code>cout <<</code>	Зумовлений потік виведення, <code><<</code> - операція лівого зсуву	<code>int c; char f; cout << "Hello"; cout << c << f;</code>	<code>ostream</code> <code>iostream</code>

Для виконання тригонометричних, логарифмічних обчислень застосовують функції C++, перелік яких подано у табл. 4.6.

Таблиця 4.6 – Деякі математичні функції

Функція	Призначення	Приклад	Бібліотека (заголовний файл)	Тип
abs	Модуль аргументу	abs(c); abs(-b);	stdlib.h	int
acos	Арккосинус	acos(0.5);	math.h або cmath	double
asin	Арксинус	asin(1);	math.h або cmath	double
atan	Арктангенс	atan(1.0);	math.h	double
cos	Косинус	cos(3.14);	math.h або cmath	double
exp	Експонента	exp(1);	math.h або cmath	double
log	Натуральний логарифм	log(exp(3));	math.h або cmath	double
log10	Десятковий логарифм	log10(100);	math.h або cmath	double
pow	Степінь числа x^y	pow(x,y);	math.h або cmath	double
sin	Синус	sin(3.14);	math.h або cmath	double
sqrt	Корінь квадратний	sqrt(144);	math.h або cmath	double
tan	Тангенс	tan(3.14);	math.h або cmath	double

Особливості використання функцій введення та виведення

1. При введенні та виведенні даних за допомогою класів потоків з рядка, який має символи пропусків, вводяться та виводяться символи тільки до першого символу пропуску.
2. При введенні даних потоком використовується стандартний потік вводу `stdin`, при виведенні поточкових даних використовується стандартний потік виводу `stdout`.
3. Після введення символу за допомогою функцій `getchar()`, `getche()`, `_getch()` необхідно очищати буфер клавіатури, інакше наступний ввід даних буде неможливий (необхідно опрацювати введення символу ENTER, використовуючи функцію `fflush(stdin)`) або `cin.get()`
4. При введенні символу без відображення його на екрані натискувати клавішу ENTER після символу непотрібно.
5. Для переведення курсору на новий рядок в процесі виведення даних на екран використовується ESC-послідовність `'\n'`.
6. Для отримання кінця рядка використовується ESC-послідовність `'\0'`.
7. Для виконання табуляції курсору в процесі виведення даних на екран використовується ESC-послідовність `'\t'`.
8. При використанні функцій консольного виводу курсор не переводиться на початок наступного рядка, тому користувач повинен використовувати ESC-послідовності `'\n'` - перевід курсору на новий рядок, `'\r'` - перевід курсору на початок рядка.

4.1.3 Приклад виконання

Розглянемо виконання першого інкременту (лабораторної роботи) за проєктом «Програмна система конвертації одиниць вимірювання»

№	Тема лаб. роботи	Що додається до проєкту	Результат інкременту
1	Базові обчислення та введення/виведення	Введення значення та вибір одиниць вимірювання. Реалізація простих конвертацій (метри–кілометри, грами–кілограми). Виведення результату	Програма виконує базову конвертацію одиниць вимірювання

Вимоги до написання коду

Код першого інкременту має демонструвати лінійну послідовність основних етапів обробки даних:

введення початкових даних → обчислення результату → виведення результату.

У програмі необхідно:

- підключити заголовний файл `iostream`;
- реалізувати точку входу `main()`;
- використовувати тип `int` для номера обраної операції;
- використовувати тип `double` для початкового значення та результату конвертації;
- ініціалізувати змінні до їх першого використання;
- застосувати `std::cin` і `std::cout` або їх скорочений запис за умови використання `using namespace std`;
- записати формули конвертації у вигляді арифметичних виразів C++;
- використовувати дійсні числові літерали, наприклад `1000.0`, щоб уникнути небажаного цілочисельного ділення;
- виводити результат разом із назвою або позначенням отриманої одиниці вимірювання;
- використовувати змістовні назви змінних;
- додати короткі коментарі до змінних і основних етапів обчислення;
- дотримуватися єдиного стилю відступів і розташування фігурних дужок;
- завершити функцію `main()` оператором `return 0`.

На першому етапі програма може бути реалізована в межах функції `main()`, оскільки метою інкременту є опанування типів даних, введення, виведення та обчислення виразів. Декомпозиція програми на окремі функції вводиться в наступному інкременті.

Для позначення коефіцієнтів конвертації доцільно застосовувати іменовані константи:

```
const double METERS_IN_KILOMETER = 1000.0;
const double GRAMS_IN_KILOGRAM = 1000.0;
```

Це робить призначення числових значень зрозумілим і спрощує подальшу модифікацію програми.

Програма повинна містити такі логічні блоки:

1. Початок роботи програми.

2. Оголошення та ініціалізація змінних.
3. Виведення назви програми.
4. Введення початкового значення.
5. Введення номера конвертації.
6. Обчислення результату.
7. Виведення результату.
8. Завершення програми.

Нижче (рис. 4.1) подана **HIPO** (**H**ierarchical **I**nterface **P**rocess **O**utput) діаграма (ієрархія Вхід – Процес – Вихід), яка демонструє структуру програми у вигляді ієрархії модулів (функцій) програми та їх підпорядкованість.

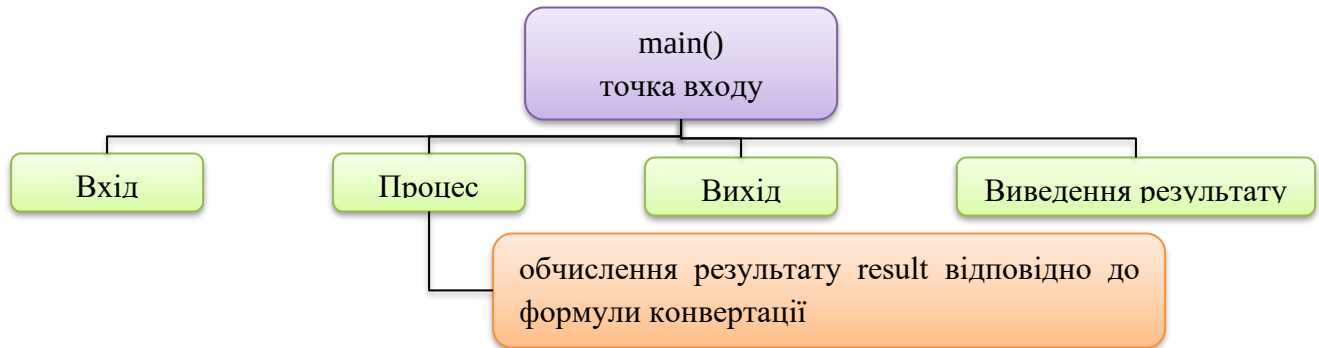


Рис. 4.1 – HIPO діаграма інкременту 1 програми конвертації одиниць вимірювання

Приклад коду та результату роботи

```

/*
 * Проєкт:    Програмна система конвертації одиниць вимірювання
 * Інкремент: 1 – типи даних, введення/виведення, обчислення виразів
 *
 * ДОДАНО В ЦЬОМУ ІНКРЕМЕНТІ:
 * - введення числового значення і номера операції;
 * - арифметичні вирази для базових перетворень;
 * - виведення результату з позначенням одиниці вимірювання.
 *
 * НАВЧАЛЬНЕ СПРОЩЕННЯ:
 * Уся логіка розміщена у main(), щоб зосередити увагу на типах даних,
 * потоках введення/виведення та арифметичних виразах. Розгалуження
 * використано як готовий диспетчер операцій і детально опрацьовується
 * в інкременті 2.
 *
 * ПЛАНОВИЙ РЕФАКТОРИНГ:
 * В інкременті 2 введення, меню та конвертації буде винесено в окремі
 * функції; буде додано перевірку коректності даних і категорії операцій.
 */

#include <locale>
#include <iostream>

int main()
{
    std::setlocale(LC_ALL, "");

    constexpr double UNITS_PER_KILO_UNIT = 1000.0;

    double inputValue = 0.0;
  
```

```

int operation = 0;

std::cout << "=====\n";
std::cout << "  КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ\n";
std::cout << "=====\n";
std::cout << "Введіть значення: ";

if (!(std::cin >> inputValue))
{
    std::cout << "Помилка: потрібно ввести число.\n";
    return 1;
}

std::cout << "Оберіть конвертацію:\n";
std::cout << "1 - Метри -> кілометри\n";
std::cout << "2 - Кілометри -> метри\n";
std::cout << "3 - Грами -> кілограми\n";
std::cout << "4 - Кілограми -> грами\n";
std::cout << "Ваш вибір: ";

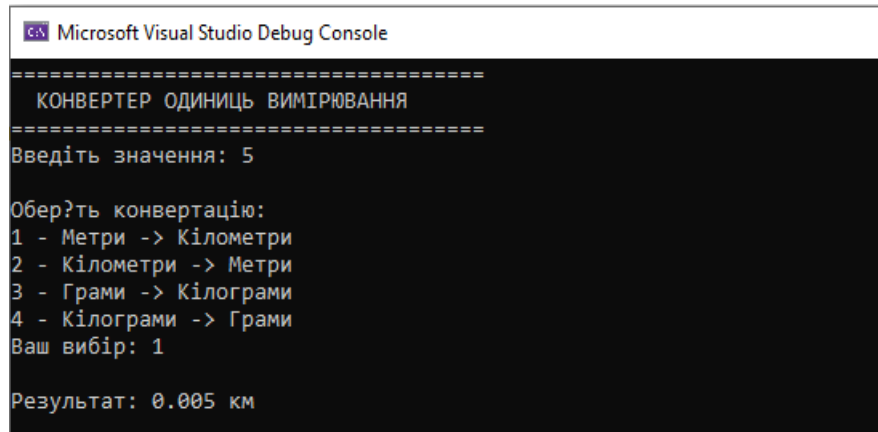
if (!(std::cin >> operation))
{
    std::cout << "Помилка: номер операції має бути цілим числом.\n";
    return 1;
}

double result = 0.0;

if (operation == 1)
{
    result = inputValue / UNITS_PER_KILO_UNIT;
    std::cout << "Результат: " << result << " км\n";
}
else if (operation == 2)
{
    result = inputValue * UNITS_PER_KILO_UNIT;
    std::cout << "Результат: " << result << " м\n";
}
else if (operation == 3)
{
    result = inputValue / UNITS_PER_KILO_UNIT;
    std::cout << "Результат: " << result << " кг\n";
}
else if (operation == 4)
{
    result = inputValue * UNITS_PER_KILO_UNIT;
    std::cout << "Результат: " << result << " г\n";
}
else
{
    std::cout << "Помилка: операції з таким номером немає.\n";
    return 1;
}

return 0;
}

```



```
Microsoft Visual Studio Debug Console

=====
КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ
=====
Введіть значення: 5

Обер?ть конвертацію:
1 - Метри -> Кілометри
2 - Кілометри -> Метри
3 - Грами -> Кілограми
4 - Кілограми -> Грами
Ваш вибір: 1

Результат: 0.005 км
```

Рис. 4.2 Приклад роботи інкременту 1

4.1.4 Контрольні запитання

1. Як визначаються коментарі у програмному коді?
2. Яка різниця між введенням даних з консолі та із стандартного вхідного потоку?
3. Яка різниця між виведенням даних на консоль та у стандартний вихідний потік?
4. Які функції повертають значення EOF при неуспішному виконання операцій введення та виведення?
5. Які функції використовують форматний ввід та вивід? Як кодуються формати введення та виведення?
6. Які заголовні файли використовуються для визначення функцій введення та виведення даних?
7. Як можна вводити дані без відображення їх на екрані?
8. Які операції використовуються для введення та виведення даних за допомогою об'єктів класів потоків?
9. Що таке типи даних? Які вбудовані типи даних існують в мові C?
10. Які операції використовуються для обробки даних цілого, дійсного, символьного та логічного типів?

4.2 Лабораторна робота №2. Розгалужені процеси

4.2.1 Мета та завдання

Мета роботи:

1. Ознайомитися з принципами організації розгалужених обчислювальних процесів у мові C++.
2. Вивчити умовні оператори та їх використання для прийняття рішень у програмі.
3. Набути практичних навичок побудови алгоритмів з розгалуженням.
4. Засвоїти особливості логічних виразів та операцій у C++.

Завдання роботи:

1. Вивчити синтаксис та принцип роботи умовних операторів if, if-else, switch.
2. Ознайомитися з логічними операторами (&&, ||, !) та правилами їх застосування.
3. Розробити програму, що реалізує розгалужений алгоритм відповідно до заданої умови.
4. Реалізувати обробку кількох варіантів виконання програми залежно від вхідних даних.
5. Забезпечити коректне введення та виведення даних.

6. Провести тестування програми для різних наборів вхідних даних та проаналізувати результати.

4.2.2 Теоретичні відомості

Алгоритмічна конструкція, що дозволяє виконавцеві алгоритму вибрати ту чи іншу послідовність дій залежно від певних умов, називається *розгалуженням або конструкцією вибору альтернатив*.

Вибір із двох альтернатив

Алгоритмічна конструкція альтернативного розгалуження, або конструкція вибору з двох альтернатив, дозволяє виконавцеві алгоритму вибрати один із двох варіантів дій залежно від істинності деякої умови. У мовах C/C++ альтернативні розгалуження реалізуються умовним оператором (*оператором розгалуження*) і *умовним виразом*. Синтаксис умовного оператора є таким:

```
if (< умова >) <оператор1;> [else <оператор2;>]
```

Тут *if*, *else*- зарезервовані слова, <умова> – довільний логічний вираз; <оператор1;> і <оператор2;> - довільні оператори.

Виконання умовного оператора починається з обчислення значення булевого виразу <умова>. Якщо цей вираз є істинним, то виконується <оператор1> і керування передається наступному за умовним оператору (<оператор2> пропускається). Якщо вираз <умова> є хибним, то <оператор1> пропускається, а виконується лише <оператор2> і на цьому дія умовного оператора вважається завершеною.

Операція «?:» дозволяє записати умовний вираз аналогічний умовному оператору:

```
< вираз1 > ? < вираз2 > : < вираз3 >;
```

Якщо < вираз1 > приймає істинне значення, то обчислюється < вираз2 >, інакше – <вираз3>. Наприклад, умовний вираз для визначення абсолютного значення числа *x* можна записати так: *x > 0 ? abs = x : abs = -x;*

Вкладеність конструкцій вибору

Гілки деякого розгалуження можуть містити інші розгалуження. Синтаксис відповідного фрагменту C-програми такий:

```
if (<умова1>) <оператор1;>  
else if (<умова2>) <оператор2;>  
else <оператор3;>
```

Піраміди вкладених розгалужень завжди можуть бути реалізовані послідовними операторами розгалуження за рахунок ускладнення умов:

```
if (<умова1>) <оператор1;>  
if (!(<умова1>) && <умова2>) <оператор2;>  
if (!(<умова1>) && !(<умова2>)) <оператор3;>
```

Але слід зазначити, що *вкладені умовні оператори* працюють значно швидше, ніж серія умовних операторів у скороченій формі.

Операторний блок

Операторний блок, або складений оператор, - це послідовність операторів, що оточені фігурними дужками { }. Синтаксис операторного блоку має такий вигляд:

```
{   <оператор1>;  
... <оператор2>;  
}
```

Поліваріантний вибір

У мовах C/C++ алгоритмічну конструкцію поліваріантного виразу реалізовано *оператором перемикання*. Наведемо його синтаксис:

```
switch ( < селектор > )  
{   case <список констант1>: <оператор1>;   break;  
    case <список констант2>: <оператор2>;   break;  
    .....  
    default: <операторN>;   break;  
}
```

Тут **switch**, **case**, **default**, **break** - це зарезервовані слова, <селектор> - змінна або вираз, який має довільний перелічуваний тип; <список констант> - перелік розділених комами значень того самого типу, що і селектор; <оператор> - будь-який оператор; дужки «{» та «}» означають початок та кінець тіла оператора **switch**.

Оператор перемикання виконується за таким алгоритмом. Спочатку обчислюється значення виразу-селектора. Потім вибирається той список констант, до якого належить отримане значення, виконується відповідний оператор і на цьому дія оператора **switch** завершується. Якщо поточне значення селектора не збігається з жодною з констант перемикання, то виконується гілка **default**, а якщо її немає, то виконання оператора перемикання завершується завдяки наявності оператора переривання **break** у кожній гілці **case**.

4.2.3 Приклад виконання

Розглянемо виконання другого інкременту(лабораторної роботи)

2	Розгалужені обчислювальні процеси	Вибір категорії конвертації (довжина, маса, температура, час). Перевірка коректності введених даних	Інтерактивне меню вибору типу конвертації
---	-----------------------------------	---	---

Вимоги до написання коду

Код інкременту 2 має відповідати принципам **структурного програмування**. В цій програмі потрібно реалізувати базові алгоритмічні структури: *послідовність і розгалуження*. Реалізація циклів передбачена в інкременті 3. **Послідовність** реалізована через виклики функцій в **main()** в такому порядку: *налаштування середовища* → *ідентифікація автора і семантики інкременту* → *вибір категорії одиниць для конвертації* → *виконання конвертації одиниць виміру*. Цей порядок відповідає принципу *лінійної організації алгоритму*. **Розгалуження** реалізовано через оператор **if – else** при виборі категорій одиниць виміру та вибору напрямку конвертації.

Принципи **процедурного програмування** передбачають:

- декомпозицію задачі на підзадачі;
- використання функцій (процедур);
- передачу керування між ними.

Декомпозиція реалізована через розбиття програми на функції. В цій програмі з урахуванням початкового рівня підготовки студентів використані `void` функції без параметрів і функції, що повертають значення, але не мають параметрів:

- введення (`inputCategory`, `inputValue`, `inputChoice`);
- інтерфейс (`menuCategory`, ..);
- обчислення (`lengthConversion`, ...);
- сервісні (`environmentSetup`, `authorIdentification`).

Побудова функцій має враховувати **принцип єдиної відповідальності**: кожна функція повинна виконувати *одну чітко визначену задачу*. Тому для відображення меню, введення даних з консолі, виконання розрахунків використовуються окремі функції.

Нижче (рис. 4.3) подана **HIPO (Hierarchical Input Process Output)** діаграма (ієрархія Вхід – Процес – Вихід), яка демонструє структуру програми у вигляді ієрархії функцій програми та їх підпорядкованість.

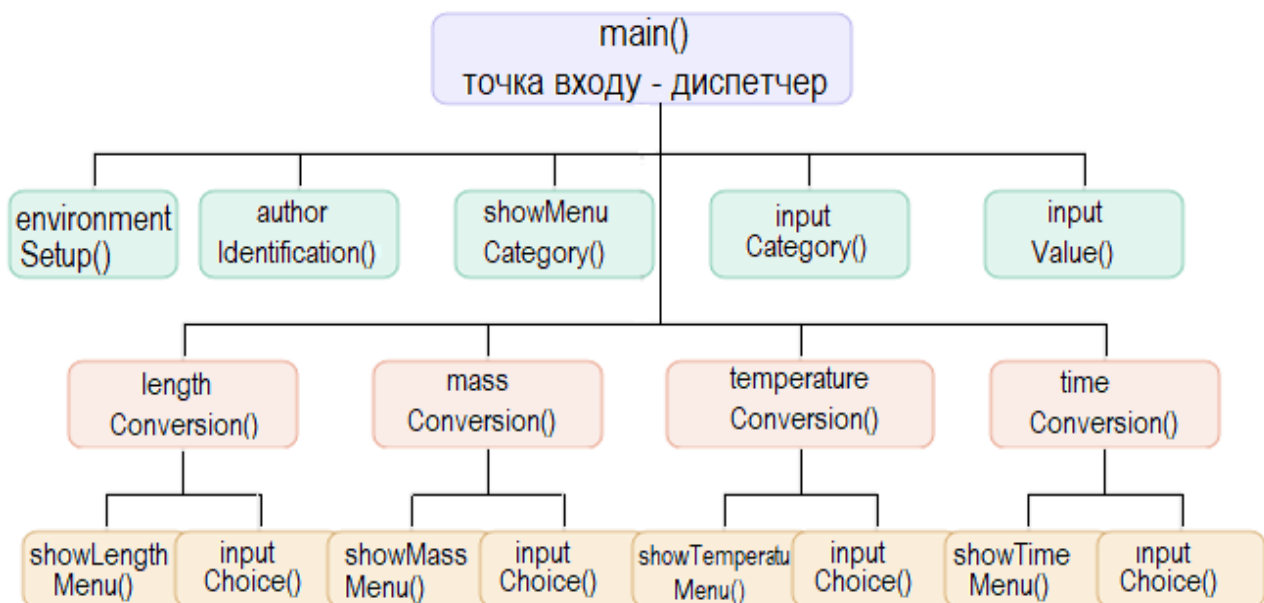


Рисунок 4.3 HIPO діаграма інкременту 2 програми конвертації одиниць вимірювання

Приклад коду та результату роботи

```

/*
* Проєкт:    Програмна система конвертації одиниць вимірювання
* Інкремент: 2 – розгалужені обчислювальні процеси
*
* ДОДАНО В ЦЬОМУ ІНКРЕМЕНТІ:
* – категорії «довжина», «маса», «температура» і «час»;
* – вибір гілки обчислень операторами switch та if;
* – перевірку типу і діапазону введених даних;
* – декомпозицію програми на невеликі функції.
*
* РЕФАКТОРИНГ ПОПЕРЕДНЬОГО ІНКРЕМЕНТУ:
* – main() перетворено на короткий диспетчер;
* – усунуто дублювання перевірки потоку введення;
* – числові коефіцієнти оформлено як іменовані константи;
* – глобальні змінні не використовуються.

```

```

*
* НАВЧАЛЬНЕ СПРОЩЕННЯ:
* Меню виконується один раз. Допоміжні функції читання використовують
* параметри-посилання як готовий інфраструктурний засіб; детально посилання
* розглядаються в інкременті 8.
*
* ПЛАНОВИЙ РЕФАКТОРИНГ:
* В інкременті 3 буде додано цикл повторення програми та табулювання
* конвертацій у заданому діапазоні.
*/

#include <clocale>
#include <iostream>
#include <limits>

namespace
{
    constexpr double UNITS_PER_KILO_UNIT = 1000.0;
    constexpr double SECONDS_PER_MINUTE = 60.0;
    constexpr double CELSIUS_TO_FAHRENHEIT_SCALE = 9.0 / 5.0;
    constexpr double FAHRENHEIT_OFFSET = 32.0;
    constexpr int MIN_CATEGORY = 1;
    constexpr int MAX_CATEGORY = 4;
    constexpr int MIN_DIRECTION = 1;
    constexpr int MAX_DIRECTION = 2;

    void clearInputStream()
    {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }

    bool readInt(const char prompt[], int& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }

        std::cout << "Помилка: потрібно ввести ціле число.\n";
        clearInputStream();
        return false;
    }

    bool readDouble(const char prompt[], double& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }

        std::cout << "Помилка: потрібно ввести число.\n";
        clearInputStream();
        return false;
    }

    bool isInRange(int value, int minimum, int maximum)
    {
        return value >= minimum && value <= maximum;
    }
}

```

```

}

void printProgramHeader()
{
    std::cout << "=====\n";
    std::cout << "    КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ\n";
    std::cout << "=====\n";
}

void printCategoryMenu()
{
    std::cout << "Оберіть категорію:\n";
    std::cout << "1 - Довжина\n";
    std::cout << "2 - Маса\n";
    std::cout << "3 - Температура\n";
    std::cout << "4 - Час\n";
}

void printDirectionMenu(const char firstDirection[], const char secondDirection[])
{
    std::cout << "1 - " << firstDirection << '\n';
    std::cout << "2 - " << secondDirection << '\n';
}

void printResult(double result, const char unit[])
{
    std::cout << "Результат: " << result << ' ' << unit << '\n';
}

void convertLength(double value)
{
    printDirectionMenu("Метри -> кілометри", "Кілометри -> метри");

    int direction = 0;
    if (!readInt("Оберіть напрям: ", direction) ||
        !isInRange(direction, MIN_DIRECTION, MAX_DIRECTION))
    {
        std::cout << "Помилка: невірний напрям конвертації.\n";
        return;
    }

    if (direction == 1)
    {
        printResult(value / UNITS_PER_KILO_UNIT, "км");
    }
    else
    {
        printResult(value * UNITS_PER_KILO_UNIT, "м");
    }
}

void convertMass(double value)
{
    printDirectionMenu("Грами -> кілограми", "Кілограми -> грами");

    int direction = 0;
    if (!readInt("Оберіть напрям: ", direction) ||
        !isInRange(direction, MIN_DIRECTION, MAX_DIRECTION))
    {
        std::cout << "Помилка: невірний напрям конвертації.\n";
        return;
    }
}

```

```

    }

    if (direction == 1)
    {
        printResult(value / UNITS_PER_KILO_UNIT, "кг");
    }
    else
    {
        printResult(value * UNITS_PER_KILO_UNIT, "г");
    }
}

void convertTemperature(double value)
{
    printDirectionMenu("Цельсій -> Фаренгейт", "Фаренгейт -> Цельсій");

    int direction = 0;
    if (!readInt("Оберіть напрям: ", direction) ||
        !isInRange(direction, MIN_DIRECTION, MAX_DIRECTION))
    {
        std::cout << "Помилка: невірний напрям конвертації.\n";
        return;
    }

    if (direction == 1)
    {
        printResult(value * CELSIUS_TO_FAHRENHEIT_SCALE + FAHRENHEIT_OFFSET, "°F");
    }
    else
    {
        printResult((value - FAHRENHEIT_OFFSET) / CELSIUS_TO_FAHRENHEIT_SCALE, "°C");
    }
}

void convertTime(double value)
{
    printDirectionMenu("Секунди -> хвилини", "Хвилини -> секунди");

    int direction = 0;
    if (!readInt("Оберіть напрям: ", direction) ||
        !isInRange(direction, MIN_DIRECTION, MAX_DIRECTION))
    {
        std::cout << "Помилка: невірний напрям конвертації.\n";
        return;
    }

    if (direction == 1)
    {
        printResult(value / SECONDS_PER_MINUTE, "хв");
    }
    else
    {
        printResult(value * SECONDS_PER_MINUTE, "с");
    }
}

int runApplication()
{
    printProgramHeader();
    printCategoryMenu();
}

```

```

int category = 0;
if (!readInt("Вкажіть номер категорії: ", category) ||
    !isInRange(category, MIN_CATEGORY, MAX_CATEGORY))
{
    std::cout << "Помилка: категорія має бути в діапазоні 1..4.\n";
    return 1;
}

double value = 0.0;
if (!readDouble("Введіть значення для конвертації: ", value))
{
    return 1;
}

switch (category)
{
case 1:
    convertLength(value);
    break;
case 2:
    convertMass(value);
    break;
case 3:
    convertTemperature(value);
    break;
case 4:
    convertTime(value);
    break;
default:
    return 1;
}

return 0;
}
} // namespace

int main()
{
    std::setlocale(LC_ALL, "");
    return runApplication();
}

```

```

Консоль отладки Microsoft Visual Studio
Проект # назва...
Інкремент 2: умова...
Автор Студент XXX, група ІПЗ-16
=====
    КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ
=====
Оберіть категорію:
1 - Довжина
2 - Маса
3 - Температура
4 - Час
Вкажіть номер категорії: 1
Введіть значення для конвертації: 45
Конвертація одиниць вимірювання довжини:
1 - Метри -> Кілометри
2 - Кілометри -> Метри
Ваш вибір: Вкажіть напрям конвертації: 2
Результат: 45000 м

```

Рис. 4.4 – Приклад роботи інкременту 2

4.2.4 Контрольні запитання

1. Як формується складена умова, що об'єднує декілька простих умов?
2. Наведіть синтаксис оператора одноальтернативного розгалуження.
3. До яких типів даних не може належати значення виразу-селектора в операторі вибору?
4. Яка структура працює ефективніше: вкладені оператори `if...else` чи серія операторів `if`? Відповідь обґрунтуйте.
5. Чи можна перервати роботу програми за допомогою оператора `break`?
6. Чому в гілках оператора `switch` слід використовувати оператор `break`?
7. Що таке умовний вираз і як він записується?
8. Що працює ефективніше: умовний вираз чи оператор `if...else`?
9. Чи є конструкція `(a > b) ? a : b`; абсолютно ідентичною до оператора `if (a > b) return a; else return b`; з точки зору результату виразу?
10. Чи можна використовувати тип `double` як вираз у `switch`?

4.3 Лабораторна робота №3. Циклічні обчислювальні процеси

4.3.1 Мета та завдання

Мета роботи:

1. Ознайомитися з принципами організації циклічних обчислювальних процесів у мові C++.
2. Вивчити основні види циклів та їх застосування для реалізації повторюваних дій.
3. Набути практичних навичок розробки алгоритмів з використанням циклів.
4. Засвоїти особливості керування ітераційними процесами та умовами їх завершення.

Завдання роботи:

1. Вивчити синтаксис і принцип роботи циклів `for`, `while`, `do-while`.
2. Ознайомитися з операторами керування циклом (`break`, `continue`).
3. Забезпечити коректне введення та виведення даних у програмі.
4. Провести тестування програми для різних вхідних даних та проаналізувати результати.

4.3.2 Теоретичні відомості

Алгоритмічна конструкція, що дозволяє виконавцеві алгоритму повторювати певну послідовність дій задану кількість разів, називається конструкцією повторення або циклічною. Кожне виконання операторів тіла циклу супроводжується перевіркою умови повторення циклу і називається його ітерацією. Якщо умова повторення істинна, то тіло циклу виконується ще раз, якщо хибна, то виконання циклу припиняється і здійснюється перехід до виконання наступного за циклом оператора. Змінні, значення яких модифікуються в тілі циклу і впливають на істинність умови повторення, називаються параметрами циклу. Виконанню будь-якого циклу має передувати ініціалізація його параметрів. У мовах C/C++ є три різновиди операторів циклу: оператор циклу з передумовою, оператор циклу з постумовою та оператор циклу з лічильником

Цикл із передумовою

У циклі з передумовою перша перевірка умови продовження циклу відбувається ще до першого виконання його тіла. Це означає, що за деяких значень параметрів циклу його тіло може

не виконатися жодного разу. Цей тип циклу застосовують лише у тому випадку, коли кількість повторень є невідомою до початку виконання циклу Синтаксис оператора циклу з передумовою:

```
while (<умова продовження циклу>
      <оператор>;
```

Тут `while (<умова продовження циклу>)` є заголовком циклу, `<оператор>` - його тілом. Тіло циклу може бути операторним блоком і містити в собі будь-які оператори: циклу, вибору, присвоєння тощо. Умова продовження циклу повинна бути виразом булевого типу.

Цикл із постумовою

Як і цикл із передумовою, *цикл із постумовою* застосовують тоді, коли кількість ітерацій циклу є невідомою до початку його виконання. Умова продовження циклу з постумовою записується після тіла циклу та вперше перевіряється після виконання операторів тіла. Цикл з постумовою за будь-яких обставин буде виконано принаймні один раз. Синтаксис оператора циклу з постумовою такий:

```
do { <оператор1>; ... <операторN>; } while ( <умова повторення циклу> );
```

Тут `do`, `while` - зарезервовані слова, `<оператор1>;...<операторN>;` - тіло циклу; `<умова повторення циклу>` - деякий булів вираз.

Цикл із лічильником

Відстежити істинність умови повторення циклу дозволяє спеціальний різновид параметра циклу, *лічильник ітерацій*. Лічильник - це змінна, що під час кожного повторення збільшується на одиницю. В операторі циклу з лічильником облік числа виконаних ітерацій вказується у заголовку, і тому цей оператор є зручною формою запису циклів із наперед визначеною кількістю повторень. Синтаксис оператора циклу з лічильником такий.

```
for ([<вираз1>]; [<вираз2>]; [<вираз3>] )
    <оператор>;
```

Тут `for` - зарезервоване слово («для»); `<вираз1>` ініціалізує лічильник та виконується один раз на початку циклу; `<вираз2>` - деякий булів вираз, котрий визначає умову повторення циклу; `<вираз3>` змінює значення лічильника циклу, найчастіше це просто операція інкремента чи декремента; `<оператор>` - простий або складений оператор, що є тілом циклу. Фраза `for()` є заголовком циклу, а зазначений після круглих дужок оператор - тілом циклу. Найчастіше `<вираз1>` записують у вигляді `<лічильник>=<початкове значення>`; `<вираз2>` є виразом відношення (`<`, `>`, `<=`, `>=`, `!=`, `==`), в якому порівнюються поточне та кінцеве значення лічильника. Початкове та кінцеве значення лічильника - це вирази того самого перелічуваного типу; лічильник - це змінна перелічуваного типу, що є узгодженим за присвоюванням з типом початкового та кінцевого значення.

Переривання та продовження циклу

У деяких програмах виникає потреба завершити цикл або його ітерацію передчасно. У мовах C/C++ цього ефекту можна досягти, використовуючи оператори `break` та `continue`.

Передчасний вихід із циклу означає, що умова повторення циклу під час виходу з нього є істинною. Для примусового переривання циклу слід виконати оператор `break`. Оператор `continue` здійснює пропуск усіх інструкцій, записаних після його виклику в тілі циклу. Таким чином, після виконання оператора `continue` завжди перевіряється умова продовження циклу.

Якщо використовуються вкладені цикли, то оператор `break` перериває той цикл, у якому він виконується. Після переривання внутрішнього циклу управління передається заголовку

зовнішнього циклу для перевірки умови його продовження. Аналогічно працює і оператор `continue`: його виконання забезпечує дії нової ітерації того циклу, в якому цей оператор здійснено.

Оператори `break` та `continue` *порушують* вимоги структурного програмування.

4.3.3 Приклад виконання

Розглянемо виконання третього інкременту(лабораторної роботи)

3	Циклічні обчислювальні процеси	Повторне виконання конвертацій. Табулювання значень (наприклад 1–10 метрів у сантиметри)	Реалізовано циклічну обробку конвертацій
---	--------------------------------	--	--

Вимоги до написання коду

Код інкременту 3 має відповідати принципам **структурного та процедурного програмування**. За вимогами структурного програмування дозволяється використовувати оператори розгалуження (`if-else`) і оператори циклу (`for`, `do-while`). За вимогами процедурного програмування потрібно зберегти декомпозицію програми на функції, що були розроблені в інкременті 2, і додати нові функції відповідно до нової функціональності. В існуючі функції потрібно внести зміни відповідно до модифікації алгоритмів.

В цій програмі з урахуванням початкового рівня підготовки студентів рекомендовано використати `void` функції без параметрів і функції, що повертають значення, але не мають параметрів.

Побудова функцій має враховувати **принцип єдиної відповідальності**: кожна функція повинна виконувати *одну чітко визначену задачу*. Тому для відображення меню, введення даних з консолі, виконання розрахунків, виведення табличних значень (табуляції значень) використовуються окремі функції.

Нижче (рис. 4.5) подана **HIPO (Hierarchical Input Process Output)** діаграма (ієрархія Вхід – Процес – Вихід), яка демонструє структуру програми у вигляді ієрархії модулів (функцій) програми та їх підпорядкованість. Діаграма демонструє додані функції: `inputStartTabParameters()`, `inputStepTabParameters()`, `printTableHeaderLength()`, `printTableHeaderMass()`, `tableLengthConversion()`, `tableMassConversion()` та їх зв'язок з існуючими функціями. В код потрібно внести доповнення у функції: `showlengthMenu()`, `showMassMenu()`, `lengthConversion()`, `massConversion()`, цикл `do-while` у функцію `main()`.

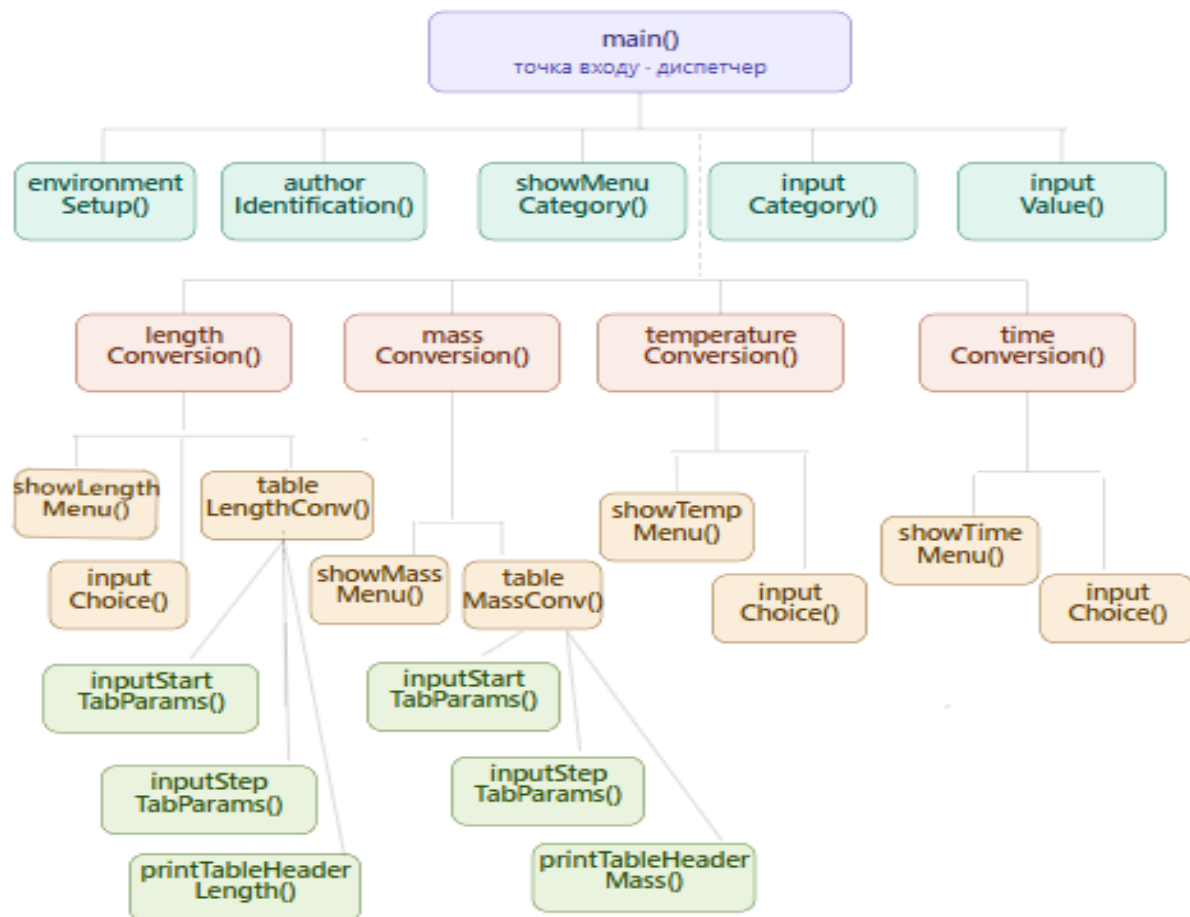


Рисунок 4.5 HIPO діаграма інкременту 3 програми конвертації одиниць вимірювання

Приклад коду та результату роботи

```

/*
* Проєкт:    Програмна система конвертації одиниць вимірювання
* Інкремент: 3 – циклічні обчислювальні процеси
*
* ДОДАНО В ЦЬОМУ ІНКРЕМЕНТІ:
* - повторення роботи програми циклом do-while;
* - побудову таблиць конвертації циклом for;
* - перевірку початкового, кінцевого значення та додатного кроку;
* - повторне введення даних після помилки.
*
* РЕФАКТОРИНГ ПОПЕРЕДНЬОГО ІНКРЕМЕНТУ:
* - цикл взаємодії винесено в runApplication();
* - спільний алгоритм таблиці використовується для довжини і маси;
* - main() залишається коротким диспетчером.
*
* НАВЧАЛЬНЕ СПРОЩЕННЯ:
* Для таблиці використовується формула current = start + index * step.
* У наступному інкременті її буде замінено рекурентним співвідношенням,
* де кожне наступне значення залежить від попереднього.
*
* ПЛАНОВИЙ РЕФАКТОРИНГ:
* В інкременті 4 буде додано рекурентне формування послідовності та
* накопичення суми результатів без дублювання табличного алгоритму.
*/

```

```

#include <locale>
#include <iomanip>
#include <iostream>
#include <limits>

namespace
{
    constexpr double UNITS_PER_KILO_UNIT = 1000.0;
    constexpr double SECONDS_PER_MINUTE = 60.0;
    constexpr double CELSIUS_TO_FAHRENHEIT_SCALE = 9.0 / 5.0;
    constexpr double FAHRENHEIT_OFFSET = 32.0;
    constexpr int MIN_CATEGORY = 1;
    constexpr int MAX_CATEGORY = 4;

    void clearInputStream()
    {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }

    bool readInt(const char prompt[], int& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }

        std::cout << "Помилка: потрібно ввести ціле число.\n";
        clearInputStream();
        return false;
    }

    bool readDouble(const char prompt[], double& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }

        std::cout << "Помилка: потрібно ввести число.\n";
        clearInputStream();
        return false;
    }

    bool readChoice(const char prompt[], int minimum, int maximum, int& value)
    {
        if (!readInt(prompt, value))
        {
            return false;
        }

        if (value < minimum || value > maximum)
        {
            std::cout << "Помилка: допустимий діапазон "
                << minimum << ".." << maximum << "...\n";
            return false;
        }
    }
}

```

```

        return true;
    }

    bool readRepeatAnswer()
    {
        char answer = 'n';
        std::cout << "Бажаєте виконати ще одну операцію? (y/n): ";
        if (!(std::cin >> answer))
        {
            clearInputStream();
            return false;
        }

        return answer == 'y' || answer == 'Y';
    }

    void printProgramHeader()
    {
        std::cout << "=====\n";
        std::cout << "  КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ\n";
        std::cout << "=====\n";
    }

    void printCategoryMenu()
    {
        std::cout << "1 - Довжина\n";
        std::cout << "2 - Маса\n";
        std::cout << "3 - Температура\n";
        std::cout << "4 - Час\n";
    }

    void printResult(double result, const char unit[])
    {
        std::cout << "Результат: " << result << ' ' << unit << '\n';
    }

    void printConversionTable(double start,
        double finish,
        double step,
        double factor,
        const char sourceUnit[],
        const char targetUnit[])
    {
        std::cout << '\n'
            << std::setw(14) << sourceUnit
            << std::setw(18) << targetUnit << '\n';
        std::cout << "-----\n";

        const int rowCount = static_cast<int>((finish - start) / step) + 1;
        for (int index = 0; index < rowCount; ++index)
        {
            const double current = start + index * step;
            std::cout << std::setw(14) << current
                << std::setw(18) << current * factor << '\n';
        }
    }

    void runTableMode(double factor,
        const char sourceUnit[],
        const char targetUnit[])
    {

```

```

double start = 0.0;
double finish = 0.0;
double step = 0.0;

if (!readDouble("Початкове значення: ", start) ||
    !readDouble("Кінцеве значення: ", finish) ||
    !readDouble("Крок: ", step))
{
    return;
}

if (finish < start)
{
    std::cout << "Помилка: кінцеве значення не може бути меншим за початкове.\n";
    return;
}

if (step <= 0.0)
{
    std::cout << "Помилка: крок має бути додатним.\n";
    return;
}

printConversionTable(start, finish, step, factor, sourceUnit, targetUnit);
}

void convertLength(double value)
{
    std::cout << "1 - Метри -> кілометри\n";
    std::cout << "2 - Кілометри -> метри\n";
    std::cout << "3 - Таблиця: метри -> кілометри\n";

    int direction = 0;
    if (!readChoice("Оберіть напрям: ", 1, 3, direction))
    {
        return;
    }

    switch (direction)
    {
    case 1:
        printResult(value / UNITS_PER_KILO_UNIT, "км");
        break;
    case 2:
        printResult(value * UNITS_PER_KILO_UNIT, "м");
        break;
    case 3:
        runTableMode(1.0 / UNITS_PER_KILO_UNIT, "метри", "кілометри");
        break;
    }
}

void convertMass(double value)
{
    std::cout << "1 - Грами -> кілограми\n";
    std::cout << "2 - Кілограми -> грами\n";
    std::cout << "3 - Таблиця: грами -> кілограми\n";

    int direction = 0;
    if (!readChoice("Оберіть напрям: ", 1, 3, direction))
    {

```

```

        return;
    }

    switch (direction)
    {
    case 1:
        printResult(value / UNITS_PER_KILO_UNIT, "кг");
        break;
    case 2:
        printResult(value * UNITS_PER_KILO_UNIT, "г");
        break;
    case 3:
        runTableMode(1.0 / UNITS_PER_KILO_UNIT, "грами", "кілограми");
        break;
    }
}

void convertTemperature(double value)
{
    std::cout << "1 - Цельсій -> Фаренгейт\n";
    std::cout << "2 - Фаренгейт -> Цельсій\n";

    int direction = 0;
    if (!readChoice("Оберіть напрям: ", 1, 2, direction))
    {
        return;
    }

    if (direction == 1)
    {
        printResult(value * CELSIUS_TO_FAHRENHEIT_SCALE + FAHRENHEIT_OFFSET, "°F");
    }
    else
    {
        printResult((value - FAHRENHEIT_OFFSET) / CELSIUS_TO_FAHRENHEIT_SCALE, "°C");
    }
}

void convertTime(double value)
{
    std::cout << "1 - Секунди -> хвилини\n";
    std::cout << "2 - Хвилини -> секунди\n";

    int direction = 0;
    if (!readChoice("Оберіть напрям: ", 1, 2, direction))
    {
        return;
    }

    if (direction == 1)
    {
        printResult(value / SECONDS_PER_MINUTE, "хв");
    }
    else
    {
        printResult(value * SECONDS_PER_MINUTE, "с");
    }
}

void runOneOperation()
{

```

```

    printProgramHeader();
    printCategoryMenu();

    int category = 0;
    if (!readChoice("Вкажіть номер категорії: ", MIN_CATEGORY, MAX_CATEGORY,
category))
    {
        return;
    }

    double value = 0.0;
    if (!readDouble("Введіть значення для конвертації: ", value))
    {
        return;
    }

    switch (category)
    {
    case 1:
        convertLength(value);
        break;
    case 2:
        convertMass(value);
        break;
    case 3:
        convertTemperature(value);
        break;
    case 4:
        convertTime(value);
        break;
    }
}

int runApplication()
{
    do
    {
        runOneOperation();
        std::cout << '\n';
    } while (readRepeatAnswer());

    return 0;
}
} // namespace

int main()
{
    std::setlocale(LC_ALL, "");
    return runApplication();
}

```

```
F:\!KNU_Shevchenko\!COURSES=DISCIPLINES\Basic Progra...
Проект # назва....
Інкремент 3: умова....
Автор Студент XXX, група ІПЗ-16
=====
    КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ
=====
Оберіть категорію:
1 - Довжина
2 - Маса
3 - Температура
4 - Час
Вкажіть номер категорії: 2
Введіть значення для конвертації: 23
Конвертація одиниці вимірювання маси:
1 - Грами -> Кілограми
2 - Кілограми -> Грами
3 - Таблиця конвертації грамів у кілограми в заданому діапазоні
Ваш вибір: Вкажіть напрям конвертації: 3
Введіть початкове значення параметра табуляції не більше 23:2
Введіть крок зміни параметра табуляції [1.. 5]: 3
Таблиця конвертації грами -> кілограми
=====
      грами      кілограми
=====
          2      0.002
          5      0.005
          8      0.008
         11      0.011
         14      0.014
         17      0.017
         20      0.02
         23      0.023
Бажаєте повторити? (y/n): n
```

Рис. 4.6 – Приклад роботи інкременту 3

4.3.4 Контрольні запитання

1. У чому полягає відмінність між циклами з передумовою та циклами з постумовою?
2. Якому типу даних може належати лічильник у циклі `for`?
3. Яке значення має лічильник після завершення циклу `for`?
4. Що може спричинити «зациклення» програми?
5. За яких умов цикли `while` та `for` не виконаються жодного разу?
6. Коли цикл виконується лише один раз?
7. У чому полягає відмінність між такими операторами циклів, як `for`, `while`, `do ... while`?
8. Поясніть алгоритм роботи вкладених циклів.
9. Якщо оператор `break` розміщується у внутрішньому циклі, то який цикл він перериває?
10. За яких умов слід обирати цикли `for` та `while`?

4.4 Лабораторна робота №4. Рекурентні співвідношення

4.4.1 Мета та завдання

Мета роботи:

1. Ознайомитися з поняттям рекурентних співвідношень та їх роллю в алгоритмах.
2. Вивчити принципи побудови рекурсивних алгоритмів у мові C++.
3. Набути практичних навичок реалізації рекурсії для обчислення послідовностей.
4. Засвоїти відмінності між рекурсивними та ітераційними підходами до розв'язання задач.

Завдання роботи:

1. Вивчити поняття рекурентного співвідношення та базового випадку.
2. Ознайомитися з синтаксисом і принципом роботи рекурсивних функцій у C++.
3. Реалізувати перевірку коректності вхідних даних та граничних умов.
4. Порівняти рекурсивну та ітераційну реалізації (за можливості).
5. Провести тестування програми та проаналізувати ефективність реалізації.

4.4.2 Теоретичні відомості

Рекурентні співвідношення

Формула, що виражає член послідовності через один або декілька попередніх, називається *рекурентним співвідношенням*. Послідовність, члени якої задовольняють деякому рекурентному співвідношенню, називається *рекурентною*.

У загальному випадку рекурентне співвідношення визначає залежність члена послідовності $\{S_n\}$ від k попередніх:

$$S_n = F(S_{n-k}, \dots, S_{n-1})$$

Наближене значення суми ряду можна отримати або обмежуючись сумою перших n його членів, або обчислюючи суму з наперед заданою точністю. Формула загального члена даного ряду є достатньо простою, але використовувати її не раціонально, оскільки для кожного члена ряду треба обчислювати степінь і факторіал. Набагато вищої ефективності можна досягти, обчислюючи член ряду за допомогою рекурентного співвідношення. Найпростішими прикладами рекурентних послідовностей є арифметична та геометрична прогресії, елементи яких пов'язані з попередніми елементами співвідношеннями $a_n = a_{n-1} + d$ та $a_n = a_{n-1} \times q$, де d та q - деякі сталі величини, a_n - значення елемента ряду на кроці n .

Із заданою точністю може бути обчислена сума лише збіжного ряду, а довільний степеневий ряд має певну область збіжності (можливо, порожню), тобто збігається не за всіх, а лише за деяких значень x (ряд, що розглядається нами як приклад, збігається для будь-якого дійсного x). По-друге, простий спосіб перевірки точності часткової суми ряду існує не для всіх рядів. Такий спосіб існує, зокрема, для знакопереміжних рядів, абсолютні величини членів яких, починаючи з деякого номера, утворюють монотонно спадну послідовність. Для таких рядів сума всіх членів, починаючи від $(n+1)$ -го, є меншою за модулем від n -го

$$|a_n| > \left| \sum_{i=n+1}^{\infty} a_i \right|$$

Функції користувача

Функція користувача містить оператори, що логічно не залежать від іншого коду програми, і може викликатися з будь-якого місця програми довільну кількість разів. Формат функції визначає її заголовок та тіло, яке містить оператори відповідно до алгоритму. Визначення функції передбачає наявність заголовка та тіла. Функцію без заданого прототипу визначають перед функціями, які її викликають, а ту, що має прототип, можна визначати в будь-якому місці програми. Функція має повернути тільки одне визначене значення до основної програми. Тип значення, що повертається, визначається у заголовку функції.

Оголошення функції (прототип) задається у вигляді:

```
тип_значення_що_повертається ім'я_функції  
(тип_параметра ім'я_параметра1, тип_параметра ім'я_параметра2);
```

Визначення функції містить її код виконання. Заголовки функції при оголошенні та визначенні збігаються. Не ставиться символ ; наприкінці заголовка. Для повернення значення з функції в тілі функції необхідно мати оператор повернення **return**.

Приклад визначення найбільшого з двох чисел:

```
float max(float a, float b) //заголовок функції з параметрами  
{  
    float m; //найбільше значення  
    m = (a > b) ? a : b; //умовний вираз  
    return m; //значення, що визначилося та повертається  
}
```

Функцію можна оголосити у вигляді:

```
void ім'я_функції(void);
```

Тип **void** значення, що повертається, означає, що така функція не повертає ніякого значення. Оператор **return** не потрібний. Тип **void** у списку параметрів означає, що функція не має параметрів.

Функція виконує задовані в ній дії за допомогою оператора виклику. У процесі визначення функції у заголовку описуються формальні параметри, які замінюються на аргументи під час виклику функції. Якщо функція не повертає значення у точку виклику, формат оператора її виклику має такий вигляд:

```
ім'я_функції(аргументи);
```

Якщо функція повертає будь-яке значення, то під час її виклику це значення треба присвоїти змінній відповідного типу. Формат оператора виклику такий:

```
ім'я_змінної = ім'я_функції(аргументи);
```

4.4.3 Приклад виконання

Розглянемо виконання четвертого інкременту(лабораторної роботи)

4	Рекурентні співвідношення	Покрокова конвертація через проміжні одиниці (мм→см→м→км). Накопичувальні перетворення	Додано багатокрокову конвертацію одиниць
---	---------------------------	---	--

Вимоги до написання коду

Код інкременту 4 має відповідати принципам **структурного та процедурного програмування**. За вимогами структурного програмування дозволяється використовувати оператори розгалуження (**if-else**, **switch**) і оператори циклу (**for**, **do-while**, **while**). За вимогами процедурного програмування потрібно зберегти декомпозицію програми на функції, що були розроблені в інкрементах 2, 3 і додати нові функції відповідно до нової функціональності. В існуючі функції потрібно внести зміни відповідно до модифікації алгоритмів.

В цій програмі рекомендовано використати як **void** функції без параметрів, так і функції, що повертають значення, але не мають параметрів, і функції, що мають параметри і повертають значення. В оновленій версії програми слід провести **рефакторинг**, у ході якого глобальні змінні замінити на локальні з подальшою передачею їх як **аргументів функцій**. Це дозволить оптимізувати використання пам'яті та забезпечити незалежність функціональних частин коду.

Побудова функцій має враховувати **принцип єдиної відповідальності**: кожна функція повинна виконувати *одну чітко визначену задачу*. Тому для відображення меню, введення даних з консолі, виконання розрахунків, виведення табличних значень (табуляції значень) використовуються окремі функції.

Інкремент 4 передбачає реалізацію рекурентних співвідношень. Цей інкремент буде таблицею конвертації через рекурентну формулу обчислення значень параметрів табуляції: $x_0 = start$; $x_n = x_{n-1} + step$, $n = 1, 2, \dots, N$, де N – кількість кроків від *start* до *finish*, x_0 - перший член рекурентного ряду. Кожне наступне значення параметру табуляції обчислюється через попереднє, а не через незалежний лічильник циклу.

Додатково обчислюється накопичена сума результатів конвертації за рекурентним співвідношенням: $S_0 = y_0$; $S_n = S_{n-1} + y_n$; $y_n = x_n * factor$, де $y_i, i = 0, \dots, N$ – результат конвертації поточного кроку таблиці, *factor* – коефіцієнт лінійної конвертації: $result = argument * factor$.

На рисунку 4.7 подана НІРО діаграма інкременту 4. Симвоом * відмічені функції, які модифіковані відповідно до функціональності інкременту 4.

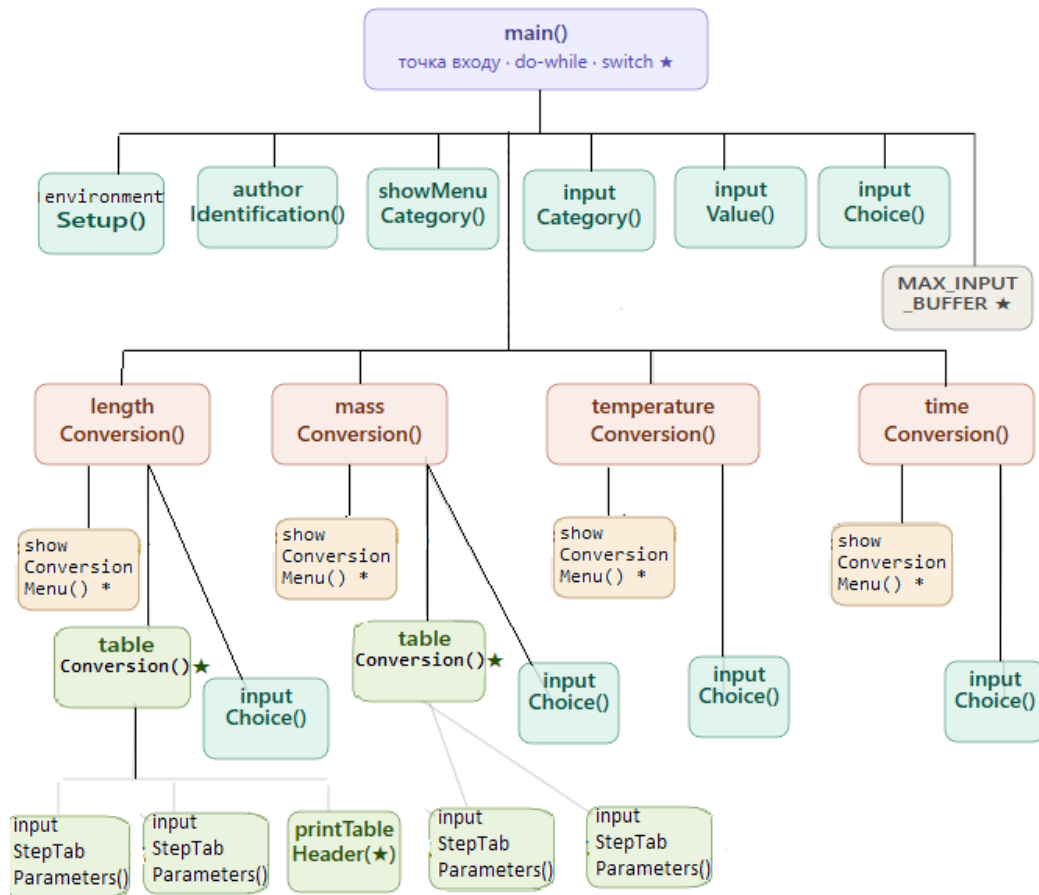


Рисунок 4.7 HIPO діаграма інкременту 4 програми конвертації одиниць вимірювання

Приклад коду та результату роботи

```

/*
* Проєкт: Програмна система конвертації одиниць вимірювання
* Інкремент: 4 – рекурентні співвідношення
*
* ДОДАНО В ЦЬОМУ ІНКРЕМЕНТІ:
* – формування послідовності  $x(n) = x(n - 1) + \text{step}$ ;
* – рекурентне накопичення суми  $S(n) = S(n - 1) + y(n)$ ;
* – єдину функцію побудови таблиць для різних категорій.
*
* РЕФАКТОРИНГ ПОПЕРЕДНЬОГО ІНКРЕМЕНТУ:
* – замість незалежного обчислення кожного аргументу використовується
* попереднє значення послідовності;
* – усі робочі дані є локальними та передаються параметрами;
* – меню, введення, обчислення і виведення розділено за SRP.
*
* НАВЧАЛЬНЕ СПРОЩЕННЯ:
* Рекурентне співвідношення реалізовано циклом. Це не рекурсія функцій:
* рекурсивний виклик буде додано в інкременті 5.
*
* ПЛАНОВИЙ РЕФАКТОРИНГ:
* В інкременті 5 буде додано рекурсивну конвертацію і рекурсивне
* виведення ланцюжка перетворень для порівняння з ітераційним варіантом.
*/

```

```
#include <locale>
```

```

#include <iomanip>
#include <iostream>
#include <limits>

namespace
{
    constexpr double UNITS_PER_KILO_UNIT = 1000.0;
    constexpr double SECONDS_PER_MINUTE = 60.0;
    constexpr double CELSIUS_TO_FAHRENHEIT_SCALE = 9.0 / 5.0;
    constexpr double FAHRENHEIT_OFFSET = 32.0;

    void clearInputStream()
    {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }

    bool readInt(const char prompt[], int& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }

        std::cout << "Помилка: потрібно ввести ціле число.\n";
        clearInputStream();
        return false;
    }

    bool readDouble(const char prompt[], double& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }

        std::cout << "Помилка: потрібно ввести число.\n";
        clearInputStream();
        return false;
    }

    bool readChoice(const char prompt[], int minimum, int maximum, int& value)
    {
        if (!readInt(prompt, value))
        {
            return false;
        }

        if (value < minimum || value > maximum)
        {
            std::cout << "Помилка: допустимий діапазон "
                      << minimum << ".." << maximum << ".\n";
            return false;
        }

        return true;
    }

    bool readRepeatAnswer()

```

```

{
    char answer = 'n';
    std::cout << "Бажаєте виконати ще одну операцію? (y/n): ";
    if (!(std::cin >> answer))
    {
        clearInputStream();
        return false;
    }
    return answer == 'y' || answer == 'Y';
}

void printProgramHeader()
{
    std::cout << "=====\n";
    std::cout << "    КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ\n";
    std::cout << "=====\n";
}

void printCategoryMenu()
{
    std::cout << "1 - Довжина\n";
    std::cout << "2 - Маса\n";
    std::cout << "3 - Температура\n";
    std::cout << "4 - Час\n";
}

void printResult(double result, const char unit[])
{
    std::cout << "Результат: " << result << ' ' << unit << '\n';
}

void printRecurrentTable(double start,
    double finish,
    double step,
    double factor,
    const char sourceUnit[],
    const char targetUnit[])
{
    std::cout << '\n'
        << std::setw(14) << sourceUnit
        << std::setw(18) << targetUnit
        << std::setw(22) << "накопичена сума" << '\n';
    std::cout << "-----\n";

    double currentArgument = start;    // x(0)
    double cumulativeSum = 0.0;        // S(-1)

    while (currentArgument <= finish)
    {
        const double currentResult = currentArgument * factor; // y(n)
        cumulativeSum += currentResult; // S(n)

        std::cout << std::setw(14) << currentArgument
            << std::setw(18) << currentResult
            << std::setw(22) << cumulativeSum << '\n';

        currentArgument += step; // x(n + 1) = x(n) + step
    }
}

void runRecurrentTable(double factor,

```

```

    const char sourceUnit[],
    const char targetUnit[])
{
    double start = 0.0;
    double finish = 0.0;
    double step = 0.0;

    if (!readDouble("Початкове значення: ", start) ||
        !readDouble("Кінцеве значення: ", finish) ||
        !readDouble("Крок: ", step))
    {
        return;
    }

    if (finish < start)
    {
        std::cout << "Помилка: кінцеве значення не може бути меншим за початкове.\n";
        return;
    }

    if (step <= 0.0)
    {
        std::cout << "Помилка: крок має бути додатним.\n";
        return;
    }

    printRecurrentTable(start, finish, step, factor, sourceUnit, targetUnit);
}

void convertLength(double value)
{
    std::cout << "1 - Метри -> кілометри\n";
    std::cout << "2 - Кілометри -> метри\n";
    std::cout << "3 - Рекурентна таблиця: метри -> кілометри\n";

    int direction = 0;
    if (!readChoice("Оберіть напрям: ", 1, 3, direction))
    {
        return;
    }

    switch (direction)
    {
    case 1:
        printResult(value / UNITS_PER_KILO_UNIT, "км");
        break;
    case 2:
        printResult(value * UNITS_PER_KILO_UNIT, "м");
        break;
    case 3:
        runRecurrentTable(1.0 / UNITS_PER_KILO_UNIT, "метри", "кілометри");
        break;
    }
}

void convertMass(double value)
{
    std::cout << "1 - Грами -> кілограми\n";
    std::cout << "2 - Кілограми -> грами\n";
    std::cout << "3 - Рекурентна таблиця: грами -> кілограми\n";
}

```

```

int direction = 0;
if (!readChoice("Оберіть напрям: ", 1, 3, direction))
{
    return;
}

switch (direction)
{
case 1:
    printResult(value / UNITS_PER_KILO_UNIT, "кг");
    break;
case 2:
    printResult(value * UNITS_PER_KILO_UNIT, "г");
    break;
case 3:
    runRecurrentTable(1.0 / UNITS_PER_KILO_UNIT, "грами", "кілограми");
    break;
}
}

void convertTemperature(double value)
{
    std::cout << "1 - Цельсій -> Фаренгейт\n";
    std::cout << "2 - Фаренгейт -> Цельсій\n";

    int direction = 0;
    if (!readChoice("Оберіть напрям: ", 1, 2, direction))
    {
        return;
    }

    if (direction == 1)
    {
        printResult(value * CELSIUS_TO_FAHRENHEIT_SCALE + FAHRENHEIT_OFFSET, "°F");
    }
    else
    {
        printResult((value - FAHRENHEIT_OFFSET) / CELSIUS_TO_FAHRENHEIT_SCALE, "°C");
    }
}

void convertTime(double value)
{
    std::cout << "1 - Секунди -> хвилини\n";
    std::cout << "2 - Хвилини -> секунди\n";

    int direction = 0;
    if (!readChoice("Оберіть напрям: ", 1, 2, direction))
    {
        return;
    }

    if (direction == 1)
    {
        printResult(value / SECONDS_PER_MINUTE, "хв");
    }
    else
    {
        printResult(value * SECONDS_PER_MINUTE, "с");
    }
}
}

```

```

void runOneOperation()
{
    printProgramHeader();
    printCategoryMenu();

    int category = 0;
    if (!readChoice("Вкажіть номер категорії: ", 1, 4, category))
    {
        return;
    }

    double value = 0.0;
    if (!readDouble("Введіть значення для конвертації: ", value))
    {
        return;
    }

    switch (category)
    {
    case 1:
        convertLength(value);
        break;
    case 2:
        convertMass(value);
        break;
    case 3:
        convertTemperature(value);
        break;
    case 4:
        convertTime(value);
        break;
    }
}

int runApplication()
{
    do
    {
        runOneOperation();
        std::cout << '\n';
    } while (readRepeatAnswer());

    return 0;
}

} // namespace

int main()
{
    std::setlocale(LC_ALL, "");
    return runApplication();
}

```

```

Консоль отладки Microsoft Visual Studio
Проект: Конвертер одиниць вимірювання
Інкремент 3: циклічні процеси, рекурентні обчислення, повтор.
Автор: Студент XXX, група ІПЗ-16
=====
      КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ
=====
Оберіть категорію:
1 - Довжина
2 - Маса
3 - Температура
4 - Час
Вкажіть номер категорії: 1
Введіть значення для конвертації: 12
Конвертація одиниць вимірювання довжини:

1 - Метри -> Кілометри
2 - Кілометри -> Метри
3 - Таблиця конвертації (рекурентний ряд) у заданому діапазоні
Вкажіть напрям конвертації: 3
Введіть початкове значення параметра табуляції (не більше 12): 3
Введіть крок зміни параметра табуляції [1..5]: 3

Конвертація одиниць вимірювання довжини
=====
      метри      кілометри
=====
          3      0.003
          6      0.006
          9      0.009
         12      0.012
=====
Накопичена сума результатів: 0.03 кілометри
Бажаєте повторити? (y/n): n

```

Рис. 4.8 – Приклад роботи інкременту 4

4.4.4 Контрольні запитання

1. Що таке ітераційний цикл?
2. Які причини зациклювання програм?
3. За яких умов цикл не виконується?
4. Яке призначення функцій C++? Яка роль функції main()?
5. Що таке прототип функції? Що таке визначення функції?
6. Дати поняття рекурсивних функцій. Технологія припинення рекурсивного виклику.
7. Чим пряма рекурсія відрізняється від непрямой?
8. Як викликається функція, що повертає значення void? Як викликається функція, що повертає значення будь-якого типу, який відрізняється від типу void?
9. Значення якого типу повертає функція, якщо тип не вказано?
10. Скільки значень може повернути функція?

4.5 Лабораторна робота №5. Рекурсивні функції

4.5.1 Мета та завдання

Мета роботи:

1. Ознайомитися з поняттям рекурсії та принципами роботи рекурсивних функцій у мові C++.
2. Вивчити структуру рекурсивної функції, зокрема базовий та рекурсивний випадки.
3. Набути практичних навичок розробки та використання рекурсивних алгоритмів.
4. Засвоїти особливості використання стеку викликів при роботі рекурсії.

Завдання роботи:

1. Вивчити синтаксис оголошення та виклику рекурсивних функцій у C++.
2. Ознайомитися з поняттям базового випадку та умовою завершення рекурсії.
3. Проаналізувати роботу рекурсивної функції на різних вхідних даних.
4. Забезпечити коректне введення та виведення даних.

5. Оцінити ефективність рекурсивного підходу та, за можливості, порівняти з ітераційною реалізацією.

4.5.2 Теоретичні відомості

Означення називається рекурсивним, якщо воно задає елементи множини за допомогою інших елементів цієї самої множини. Об'єкти, що задані рекурсивним означенням, також називаються рекурсивними. Під терміном "рекурсивно задана функція" розуміється така функція, значення якої для даного аргументу обчислюється за допомогою значень для попередніх аргументів. При кожній конкретній реалізації процес обчислення такої функції задається рекурентною послідовністю. У практиці обчислень це пов'язано з тим, що, маючи справу з реальною функцією, ми не обчислюємо її в реальному завданні на усій числовій осі, а обчислюємо для деякої кінцевої послідовності значень аргументів.

Рекурсія – це такий спосіб організації обчислювального процесу, за якого функція звертається сама до себе. Такі звернення називаються рекурсивними викликами, а функція, що містить рекурсивні виклики, - рекурсивною.

Перед тим, як записати рекурсивну функцію будь-якою мовою програмування, необхідно записати рекурентне співвідношення, що визначає метод обчислення функції. Рекурентне співвідношення повинне містити як мінімум дві умови:

- умову продовження рекурсії (крок рекурсії);
- умову закінчення рекурсії.

Розглянемо функцію $T(n)$, де аргумент n є цілим додатним числом, таким, що $n \geq 0$ або $n \geq 1$. Потрібно обчислити значення членів послідовності $t_n = T(n)$. Рекурсивне завдання послідовності t_n як послідовності значень функції $T(n)$ включає такі два етапи.

- функція $T(n)$ задається безпосередньо у вигляді числових значень для деякої кінцевої множини початкових значень аргументу $1 \leq n \leq m$;
- задається метод або формула, які дозволяють, знаючи усі значення функції $T(n)$ при $n \leq k, k \leq m$ обчислити її значення при $n = k + 1$, тобто знайти $T(k+1)$.

В результаті отримуємо рекурентні співвідношення, що описують рекурсивно задану функцію $T(n)$, рівносильну послідовності t_n .

Отриманий запис цих двох етапів

$$\begin{cases} T(1) = t_1, T(2) = t_2, \dots, T(m) = t_m \\ T(n+1) = f(t_n, t_{n-1}, \dots, t_1), \text{ при } n \geq m \end{cases}$$

називається рекурентним співвідношенням.

Рекурсію реалізують за допомогою виклику функції самої себе. При цьому в тілі функції спочатку слід перевіряти умову продовження рекурсії. Якщо вона істинна, то виходити з функції. Інакше здійснювати рекурсивний крок.

В рекурсивних підпрограмах можна виділити два процеси: рекурсивне занурення підпрограми у себе, що відбувається доти, доки параметр не сягне граничного значення, та рекурсивне повернення з підпрограми, що відбувається доти, доки параметр не сягне

початкового значення. Рекурсія реалізує нескінченний цикл, отже застосовувати її слід тільки у випадках, коли без неї неможливо обійтися.

Розглянемо рекурсивну функцію, що обчислює n -й член послідовності чисел Фібоначчі. Ці числа визначаються рекурентним співвідношенням:

$$\begin{cases} f_0 = 0, & n = 0 \\ f_1 = 1, & n = 1 \\ f_n = f_{n-1} + f_{n-2}, & n > 1 \end{cases}$$

Програмний код, що реалізує рекурсивні обчислення, такий:

```
int Fib(int n)
{
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else return Fib(n-1)+Fib(n-2);
}
```

4.5.3 Приклад виконання

Розглянемо виконання п'ятого інкременту(лабораторної роботи).

5	Рекурсивні функції	Рекурсивна конвертація через ланцюжок одиниць. Рекурсивний пошук потрібного перетворення	Рекурсивна система перетворення одиниць
---	--------------------	--	---

Вимоги до написання коду

Код інкременту 5 має відповідати принципам **структурного та процедурного програмування**. За вимогами процедурного програмування потрібно зберегти декомпозицію програми на функції, що були розроблені в інкрементах 2, 3 і 4, і додати нові функції відповідно до нової функціональності, пов'язаною з рекурсивними обчисленнями. В існуючі функції потрібно внести зміни відповідно до модифікації алгоритмів.

В цій програмі рекомендовано використати усі типи функцій C++: **void** функції без параметрів, **void** функції з параметрами, функції, що повертають значення та мають параметри. Побудова функцій має враховувати **принцип єдиної відповідальності**: кожна функція повинна виконувати *одну чітко визначену задачу*. Тому для відображення меню, введення даних з консолі, виконання розрахунків, виведення табличних значень (табуляції значень) використовуються окремі функції.

Інкремент 5 передбачає реалізацію рекурсивних функцій для визначення коефіцієнта конвертації для конкретного кроку, для конвертації одиниці виміру довжини, для виведення таблиці із відображенням усіх кроків конвертації, для підрахунку кроків конвертації.

На рисунку 4.9 подана узагальнена HIPO діаграма інкременту 5.

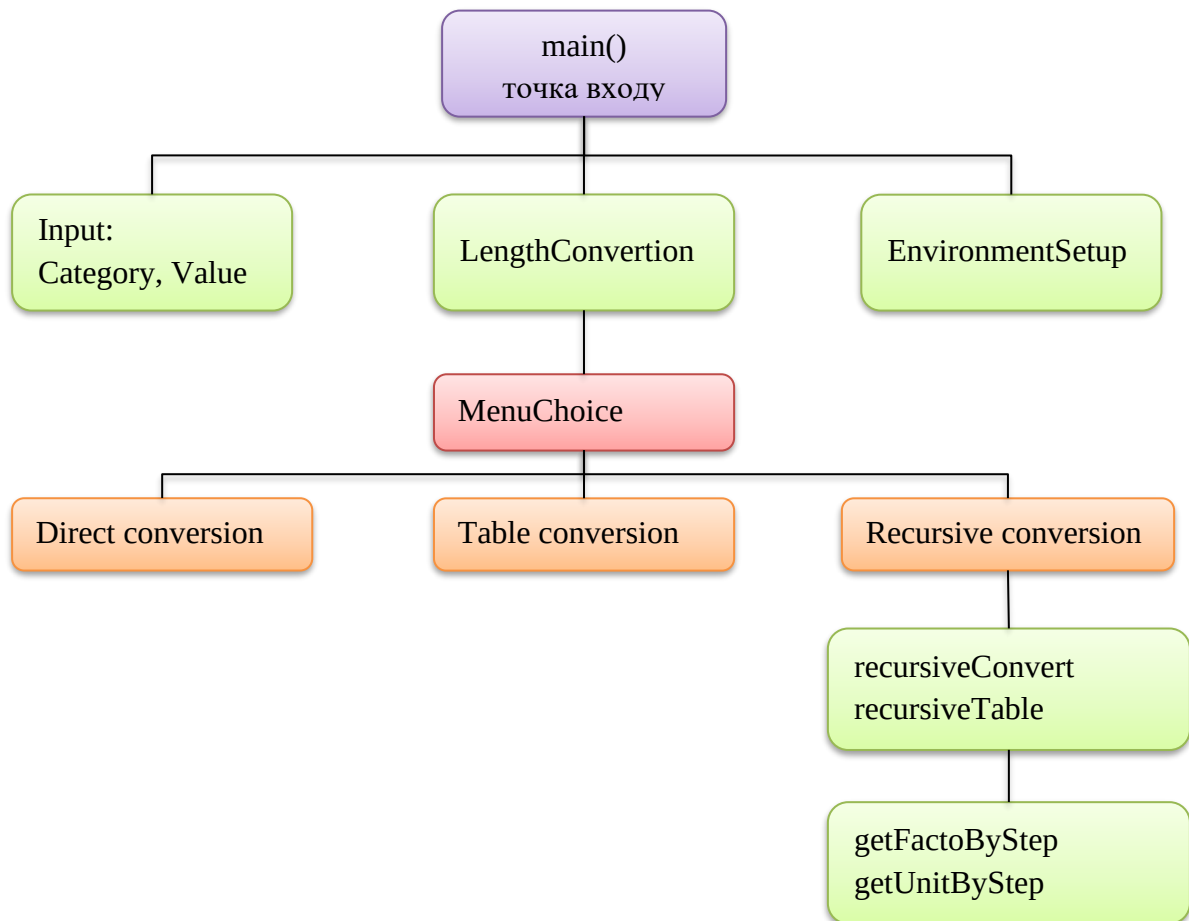


Рис. 4.9 НІРО діаграма інкременту 5 (узагальнена)

Приклад коду та результату роботи

```

/*
* Проєкт: Програмна система конвертації одиниць вимірювання
* Інкремент: 5 – рекурсивні функції
*
* ДОДАНО В ЦЬОМУ ІНКРЕМЕНТІ:
* – рекурсивну конвертацію ланцюжком мм -> см -> м -> км;
* – рекурсивне виведення всіх проміжних кроків;
* – явний базовий випадок і рекурсивний крок;
* – порівняння рекурсивного та ітераційного підходів.
*
* РЕФАКТОРИНГ ПОПЕРЕДНЬОГО ІНКРЕМЕНТУ:
* – коефіцієнт і назву одиниці для кожного кроку отримують окремі функції;
* – рекурсивна функція виконує лише обчислення, а функція виведення – лише
* формування навчального ланцюжка (SRP).
*
* НАВЧАЛЬНЕ СПРОЩЕННЯ:
* Кількість кроків і послідовність одиниць фіксовані. Рекурсія тут має
* навчальний характер; для простої конвертації ітераційний варіант коротший.
*
* ПЛАНОВИЙ РЕФАКТОРИНГ:
* В інкременті 6 фіксовану послідовність буде подано одновимірними масивами,
* що дасть змогу працювати з довільними парами підтримуваних одиниць.
*/

```

```

#include <locale>
#include <iomanip>
#include <iostream>
#include <limits>

namespace
{
    constexpr double UNITS_PER_KILO_UNIT = 1000.0;
    constexpr double SECONDS_PER_MINUTE = 60.0;
    constexpr double CELSIUS_TO_FAHRENHEIT_SCALE = 9.0 / 5.0;
    constexpr double FAHRENHEIT_OFFSET = 32.0;
    constexpr int FIRST_RECURSIVE_STEP = 0;
    constexpr int RECURSIVE_STEP_COUNT = 3;

    void clearInputStream()
    {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }

    bool readInt(const char prompt[], int& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }
        std::cout << "Помилка: потрібно ввести ціле число.\n";
        clearInputStream();
        return false;
    }

    bool readDouble(const char prompt[], double& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }
        std::cout << "Помилка: потрібно ввести число.\n";
        clearInputStream();
        return false;
    }

    bool readChoice(const char prompt[], int minimum, int maximum, int& value)
    {
        if (!readInt(prompt, value))
        {
            return false;
        }
        if (value < minimum || value > maximum)
        {
            std::cout << "Помилка: допустимий діапазон "
                << minimum << ".." << maximum << ".\n";
            return false;
        }
        return true;
    }

    bool readRepeatAnswer()
    {

```

```

    char answer = 'n';
    std::cout << "Бажаєте виконати ще одну операцію? (y/n): ";
    if (!(std::cin >> answer))
    {
        clearInputStream();
        return false;
    }
    return answer == 'y' || answer == 'Y';
}

void printProgramHeader()
{
    std::cout << "=====\n";
    std::cout << "    КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ\n";
    std::cout << "=====\n";
}

void printCategoryMenu()
{
    std::cout << "1 - Довжина\n";
    std::cout << "2 - Маса\n";
    std::cout << "3 - Температура\n";
    std::cout << "4 - Час\n";
}

void printResult(double result, const char unit[])
{
    std::cout << "Результат: " << result << ' ' << unit << '\n';
}

void printRecurrentTable(double start,
    double finish,
    double step,
    double factor,
    const char sourceUnit[],
    const char targetUnit[])
{
    std::cout << std::setw(14) << sourceUnit
        << std::setw(18) << targetUnit
        << std::setw(22) << "накопичена сума" << '\n';
    std::cout << "-----\n";

    double currentArgument = start;
    double cumulativeSum = 0.0;
    while (currentArgument <= finish)
    {
        const double currentResult = currentArgument * factor;
        cumulativeSum += currentResult;
        std::cout << std::setw(14) << currentArgument
            << std::setw(18) << currentResult
            << std::setw(22) << cumulativeSum << '\n';
        currentArgument += step;
    }
}

void runRecurrentTable(double factor,
    const char sourceUnit[],
    const char targetUnit[])
{
    double start = 0.0;
    double finish = 0.0;

```

```

double step = 0.0;

if (!readDouble("Початкове значення: ", start) ||
    !readDouble("Кінцеве значення: ", finish) ||
    !readDouble("Крок: ", step))
{
    return;
}

if (finish < start || step <= 0.0)
{
    std::cout << "Помилка: перевірте межі та додатність кроку.\n";
    return;
}

printRecurrentTable(start, finish, step, factor, sourceUnit, targetUnit);
}

// === ДОДАНО В ІНКРЕМЕНТІ 5: РЕКУРСІЯ ===
double factorForRecursiveStep(int step)
{
    switch (step)
    {
        case 0:
            return 1.0 / 10.0;    // мм -> см
        case 1:
            return 1.0 / 100.0;   // см -> м
        case 2:
            return 1.0 / 1000.0;  // м -> км
        default:
            return 1.0;
    }
}

const char* unitForRecursiveStep(int step)
{
    switch (step)
    {
        case 0:
            return "мм";
        case 1:
            return "см";
        case 2:
            return "м";
        case 3:
            return "км";
        default:
            return "";
    }
}

double recursiveConvert(double value, int step)
{
    if (step == RECURSIVE_STEP_COUNT)
    {
        return value; // базовий випадок
    }

    const double nextValue = value * factorForRecursiveStep(step);
    return recursiveConvert(nextValue, step + 1); // рекурсивний крок
}

```

```

void printRecursiveChain(double value, int step)
{
    if (step == RECURSIVE_STEP_COUNT)
    {
        return; // базовий випадок
    }

    const double nextValue = value * factorForRecursiveStep(step);
    std::cout << value << ' ' << unitForRecursiveStep(step)
        << " = " << nextValue << ' '
        << unitForRecursiveStep(step + 1) << '\n';

    printRecursiveChain(nextValue, step + 1);
}

void runRecursiveDemo(double millimeters)
{
    printRecursiveChain(millimeters, FIRST_RECURSIVE_STEP);
    const double kilometers = recursiveConvert(millimeters, FIRST_RECURSIVE_STEP);
    printResult(kilometers, "км");
}

void convertLength(double value)
{
    std::cout << "1 - Метри -> кілометри\n";
    std::cout << "2 - Кілометри -> метри\n";
    std::cout << "3 - Рекурентна таблиця: метри -> кілометри\n";
    std::cout << "4 - Рекурсивний ланцюжок: мм -> км\n";

    int direction = 0;
    if (!readChoice("Оберіть напрям: ", 1, 4, direction))
    {
        return;
    }

    switch (direction)
    {
    case 1:
        printResult(value / UNITS_PER_KILO_UNIT, "км");
        break;
    case 2:
        printResult(value * UNITS_PER_KILO_UNIT, "м");
        break;
    case 3:
        runRecurrentTable(1.0 / UNITS_PER_KILO_UNIT, "метри", "кілометри");
        break;
    case 4:
        runRecursiveDemo(value);
        break;
    }
}

void convertMass(double value)
{
    std::cout << "1 - Грами -> кілограми\n";
    std::cout << "2 - Кілограми -> грами\n";
    std::cout << "3 - Рекурентна таблиця: грами -> кілограми\n";

    int direction = 0;
    if (!readChoice("Оберіть напрям: ", 1, 3, direction))

```

```

    {
        return;
    }

    switch (direction)
    {
    case 1:
        printResult(value / UNITS_PER_KILO_UNIT, "кг");
        break;
    case 2:
        printResult(value * UNITS_PER_KILO_UNIT, "г");
        break;
    case 3:
        runRecurrentTable(1.0 / UNITS_PER_KILO_UNIT, "грами", "кілограми");
        break;
    }
}

void convertTemperature(double value)
{
    std::cout << "1 - Цельсій -> Фаренгейт\n";
    std::cout << "2 - Фаренгейт -> Цельсій\n";
    int direction = 0;
    if (!readChoice("Оберіть напрям: ", 1, 2, direction))
    {
        return;
    }

    const double result = direction == 1
        ? value * CELSIUS_TO_FAHRENHEIT_SCALE + FAHRENHEIT_OFFSET
        : (value - FAHRENHEIT_OFFSET) / CELSIUS_TO_FAHRENHEIT_SCALE;
    printResult(result, direction == 1 ? "°F" : "°C");
}

void convertTime(double value)
{
    std::cout << "1 - Секунди -> хвилини\n";
    std::cout << "2 - Хвилини -> секунди\n";
    int direction = 0;
    if (!readChoice("Оберіть напрям: ", 1, 2, direction))
    {
        return;
    }

    const double result = direction == 1
        ? value / SECONDS_PER_MINUTE
        : value * SECONDS_PER_MINUTE;
    printResult(result, direction == 1 ? "хв" : "с");
}

void runOneOperation()
{
    printProgramHeader();
    printCategoryMenu();

    int category = 0;
    if (!readChoice("Вкажіть номер категорії: ", 1, 4, category))
    {
        return;
    }
}

```

```

double value = 0.0;
if (!readDouble("Введіть значення для конвертації: ", value))
{
    return;
}

switch (category)
{
case 1:
    convertLength(value);
    break;
case 2:
    convertMass(value);
    break;
case 3:
    convertTemperature(value);
    break;
case 4:
    convertTime(value);
    break;
}
}

int runApplication()
{
    do
    {
        runOneOperation();
        std::cout << '\n';
    } while (readRepeatAnswer());

    return 0;
}
} // namespace

int main()
{
    std::setlocale(LC_ALL, "");
    return runApplication();
}

```

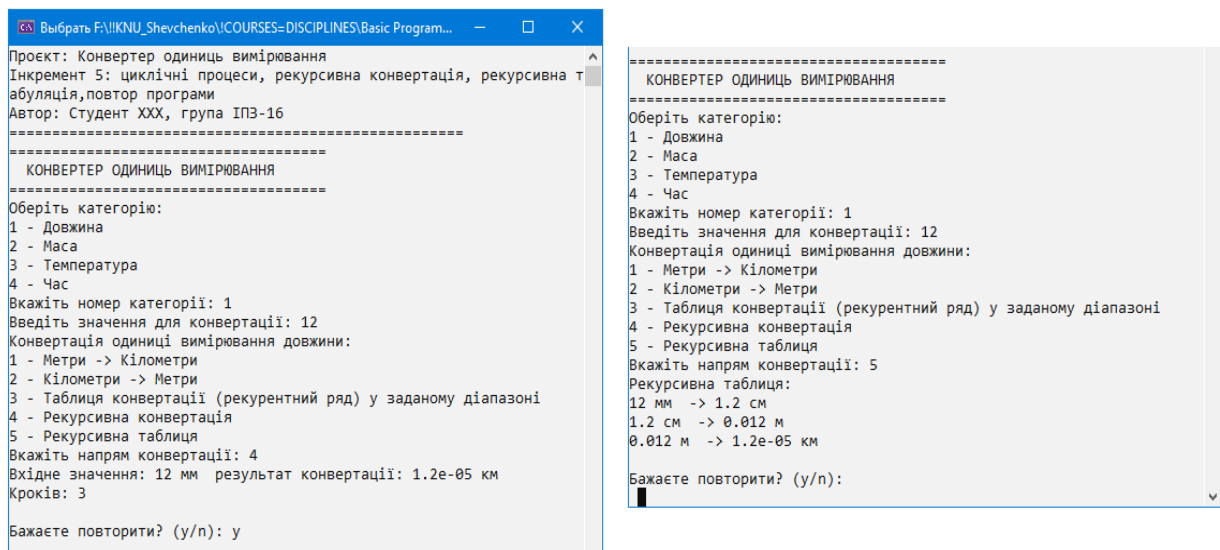


Рис. 4.10 – Приклад роботи інкременту 5

4.5.4 Контрольні запитання

1. Що таке рекурсивне визначення і рекурсивний об'єкт?
2. Визначити поняття рекурсії. Що таке рекурсивна підпрограма?
3. Як визначається глибина рекурсії?
4. Як обмежити послідовність вкладених викликів рекурсивної підпрограми?
5. Коли рекурсія неефективна і коли її необхідно уникати?
6. Чим викликається переповнення стеку під час рекурсії?
7. Чому рекурсія не може бути нескінченною навіть теоретично, у той час як цикл `while(true)` може працювати роками? Що саме закінчується в комп'ютері під час занадто глибокої рекурсії і до якої саме помилки це призводить?
8. Чому обчислення чисел Фібоначчі через класичну рекурсію $F(n) = F(n - 1) + F(n - 2)$ є надзвичайно поганою інженерною практикою для великих n ?
9. Будь-який рекурсивний алгоритм можна перетворити на ітераційний (цикл). Але якщо алгоритм за своєю природою рекурсивний (як обхід дерева чи лабіринту), що саме вам доведеться створити вручну всередині циклу, щоб замінити рекурсію?
10. Що станеться, якщо в рекурсивній функції описати «базовий випадок» (умову виходу), але логічно він ніколи не буде досягнутий під час виконання (наприклад, через помилку в кроці рекурсії)?

4.6 Лабораторна робота №6. Одновимірні масиви

4.6.1 Мета та завдання

Мета роботи:

1. Ознайомитися з поняттям одновимірного масиву та його використанням у мові C++.
2. Вивчити способи оголошення, ініціалізації та обробки масивів.
3. Набути практичних навичок роботи з елементами масиву.
4. Засвоїти методи обробки масивів за допомогою циклічних конструкцій.

Завдання роботи:

1. Вивчити синтаксис оголошення та ініціалізації одновимірних масивів.
2. Ознайомитися зі способами введення та виведення елементів масиву.
3. Реалізувати обхід масиву за допомогою циклів.
4. Виконати сортування або інші операції над елементами масиву (за потреби).
5. Провести тестування програми та проаналізувати отримані результати.

4.6.2 Теоретичні відомості

Тип масиву - це структурований тип даних, множина допустимих значень котрого складається з усіх масивів, для яких зафіксовано:

- розмірність;
- базовий тип;
- індексний тип;
- множину значень індексу.

З точки зору математики одновимірний масив - це вектор. Змінну, що матиме тип масиву, можна оголосити з використанням такого синтаксису:

<тип елементів> <ім'я масиву> [<кількість елементів>];

Основні властивості масивів:

- однорідність - усі елементи належать одному типу;
- сталість - вимірність масиву задається під час його оголошення і не змінюється протягом роботи з ним;
- рівнодоступність - спосіб доступу до всіх елементів є однаковим;
- послідовність розташування - усі елементи масиву розташовані в послідовних комірках оперативної пам'яті;
- індексованість - елементи однозначно ідентифікуються своїми індексами;
- упорядкованість індексу - індексний тип має бути простим порядковим типом даних.

Базовими операціями обробки масивів є:

- введення та виведення масиву;
- ініціалізація масиву;
- копіювання масиву;
- пошук максимального або мінімального елемента;
- обчислення узагальнювальних характеристик (сум елементів, їх добутків);
- пошук заданого елемента;
- перестановка елементів або обмін значеннями між елементами масиву;
- вставка та видалення елемента.

Базові операції обробки масивів зручно реалізовувати у вигляді функцій, що згодом можуть бути використані як «архітектурні блоки» при розв'язанні більш складних задач.

```
//ввести масив
for( int i = 0; i < n; i++)
    cin >> mas[i];
cout << "Entered array" << endl;
//вивести масив
for(int i = 0; i < n; i++)
    cout << mas[i] << " ";
cout << endl;
//генерація масиву
srand((unsigned)time( NULL )); //ініціалізація генератора
                                // випадкових чисел
for(int i = 0; i < n; ++i) //генерація псевдовипадкових чисел
    mas[i] = rand() % 30;    // в діапазоні від 0 до 29
// пошук максимального елемента
max = mark[0];                //максимальної
for(int i = 1; i < n; i++) //пошук мінімальної та максимальної оцінок
{
    if (max < mark[i]) max = mark[i]; //модифікація
                                    //поточного максимуму
}
```

```
//бульбашкове сортування
for(int i = 0; i < n - 1; i++)
    for(int j = i + 1; j < n; j++)
        if (mas[i] > mas[j])
        {
            tmp = mas[i];
            mas[i] = mas[j];
            mas[j] = tmp;
        }
```

4.6.3 Приклад виконання

Розглянемо приклад виконання шостого інкременту(лабораторної роботи).

6	Одновимірні масиви	Масив одиниць вимірювання, що підтримуються. Масив коефіцієнтів конвертації. Масив значень, що конвертуються. Операції над масивом: сортування, лінійний пошук в невідсортованому масиві, бінарний пошук у відсортованому масиві, статистика.	Каталог одиниць вимірювання і результати обробки масиву
---	--------------------	---	---

Вимоги до написання коду

У шостому інкременті необхідно розширити програму засобами роботи з одновимірними масивами. Масиви слід використовувати для зберігання підтримуваних одиниць вимірювання, коефіцієнтів конвертації та результатів обчислення кількох значень.

Під час написання програми необхідно:

1. Оголосити одновимірні масиви відповідних типів для зберігання початкових даних, коефіцієнтів конвертації та результатів обчислень.
2. Розмір масиву задавати іменованою константою, наприклад:
const int SIZE = 5;
3. Не використовувати числові значення розміру масиву безпосередньо в умовах циклів.
4. Забезпечити, щоб індекси елементів масиву належали діапазону від 0 до SIZE – 1.
5. Для введення, обробки та виведення елементів масиву застосовувати цикли.
6. Не допускати звернення до елементів за межами масиву.
7. Використовувати змістовні назви масивів і змінних, наприклад values, coefficients, results, unitsCount.
8. Реалізувати основні операції обробки масиву в окремих функціях. Доцільно передбачити функції для:
 - введення або формування масиву;
 - виведення елементів масиву;
 - обчислення результатів конвертації;
 - пошуку мінімального та максимального значень;
 - обчислення середнього значення;
 - пошуку заданого елемента;
 - сортування елементів масиву.
9. Передавати масиви у функції через параметри разом із кількістю їхніх елементів.

10. Масиви, які функція не повинна змінювати, передавати з використанням модифікатора `const`.

Приклад заголовка такої функції:

```
void printArray(const double values[], int size);
```

11. Не використовувати глобальні масиви, якщо вони не є обґрунтованими особливостями конкретного проєкту.

12. Перевіряти коректність кількості введених елементів та інших даних користувача.

13. У функції `main()` залишити організацію послідовності викликів, а основні операції над масивами винести в окремі функції.

14. Дотримуватися єдиного стилю оформлення коду, використовувати відступи та короткі коментарі до основних логічних блоків.

15. Перевірити роботу програми для типових, граничних і некоректних вхідних даних.

Результатом виконання інкременту має бути програма, яка зберігає набір значень або параметрів конвертації в одновимірних масивах, опрацьовує їх за допомогою циклів і функцій та виводить отримані результати.

Рис. 4.11 демонструє HIPO діаграму інкременту 6, на якій подані функції інкременту 5 та нові функції, що відмічені символом «*» для інкременту 6.

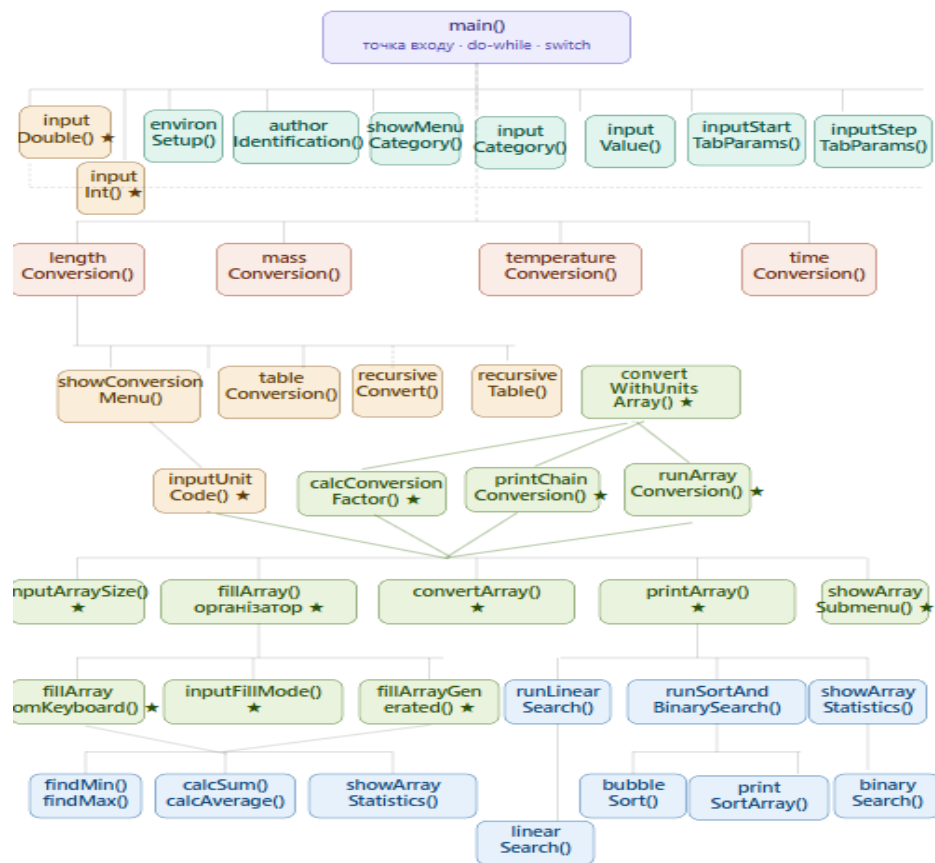


Рис. 4.11 HIPO діаграма інкременту 6

Приклад коду та результату роботи

```
/*
* Проєкт:    Програмна система конвертації одиниць вимірювання
```

```

/*
* Проект: Програмна система конвертації одиниць вимірювання
* Інкремент: 6 – одновимірні масиви
*
* ДОДАНО В ЦЬОМУ ІНКРЕМЕНТІ:
* - масив назв одиниць і масив коефіцієнтів до базової одиниці;
* - конвертацію між довільними одиницями довжини за їх індексами;
* - введення та обробку масиву значень;
* - пошук, бульбашкове сортування, бінарний пошук і статистику.
*
* РЕФАКТОРИНГ ПОПЕРЕДНЬОГО ІНКРЕМЕНТУ:
* - фіксовані switch-функції для рекурсивного ланцюжка замінено масивами;
* - коефіцієнти та назви одиниць зберігаються в одному узгодженому порядку;
* - обчислення, пошук, сортування та виведення розділено за SRP.
*
* НАВЧАЛЬНЕ СПРОЩЕННЯ:
* Використовуються статичні масиви фіксованого максимального розміру.
* Назви одиниць подані літералами const char*, без редагування тексту.
*
* ПЛАНОВИЙ РЕФАКТОРИНГ:
* В інкременті 7 коефіцієнти буде подано двовимірною матрицею;
* в інкременті 8 масиви стануть динамічними.
*/

#include <locale>
#include <cmath>
#include <iomanip>
#include <iostream>
#include <limits>

namespace
{
    constexpr int UNIT_COUNT = 5;
    constexpr int MAX_VALUE_COUNT = 20;
    constexpr double EPSILON = 1e-9;
    constexpr double SECONDS_PER_MINUTE = 60.0;
    constexpr double CELSIUS_TO_FAHRENHEIT_SCALE = 9.0 / 5.0;
    constexpr double FAHRENHEIT_OFFSET = 32.0;

    const char* const UNIT_NAMES[UNIT_COUNT] = { "мм", "см", "дм", "м", "км" };
    const double TO_MILLIMETERS[UNIT_COUNT] = { 1.0, 10.0, 100.0, 1000.0, 1000000.0 };

    void clearInputStream()
    {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }

    bool readInt(const char prompt[], int& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }
        std::cout << "Помилка: потрібно ввести ціле число.\n";
        clearInputStream();
        return false;
    }

    bool readDouble(const char prompt[], double& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {

```

```

        return true;
    }
    std::cout << "Помилка: потрібно ввести число.\n";
    clearInputStream();
    return false;
}

bool readChoice(const char prompt[], int minimum, int maximum, int& value)
{
    if (!readInt(prompt, value))
    {
        return false;
    }
    if (value < minimum || value > maximum)
    {
        std::cout << "Помилка: допустимий діапазон "
                    << minimum << ".." << maximum << ".\n";
        return false;
    }
    return true;
}

bool readRepeatAnswer()
{
    char answer = 'n';
    std::cout << "Бажаєте виконати ще одну операцію? (y/n): ";
    if (!(std::cin >> answer))
    {
        clearInputStream();
        return false;
    }
    return answer == 'y' || answer == 'Y';
}

void printHeader()
{
    std::cout << "=====\n";
    std::cout << "    КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ\n";
    std::cout << "=====\n";
}

void printResult(double result, const char unit[])
{
    std::cout << "Результат: " << result << ' ' << unit << '\n';
}

void printUnits()
{
    std::cout << "Одиниці довжини:\n";
    for (int index = 0; index < UNIT_COUNT; ++index)
    {
        std::cout << index << " - " << UNIT_NAMES[index] << '\n';
    }
}

bool readUnitIndex(const char prompt[], int& index)
{
    return readChoice(prompt, 0, UNIT_COUNT - 1, index);
}

double conversionFactor(int sourceIndex, int targetIndex)
{
    return TO_MILLIMETERS[sourceIndex] / TO_MILLIMETERS[targetIndex];
}

```

```

double convertLengthValue(double value, int sourceIndex, int targetIndex)
{
    return value * conversionFactor(sourceIndex, targetIndex);
}

void printRecurrentTable()
{
    double start = 0.0;
    double finish = 0.0;
    double step = 0.0;

    if (!readDouble("Початкове значення, м: ", start) ||
        !readDouble("Кінцеве значення, м: ", finish) ||
        !readDouble("Крок: ", step))
    {
        return;
    }
    if (finish < start || step <= 0.0)
    {
        std::cout << "Помилка: перевірте межі та крок.\n";
        return;
    }

    double current = start;
    double cumulativeSum = 0.0;
    while (current <= finish)
    {
        const double kilometers = current / 1000.0;
        cumulativeSum += kilometers;
        std::cout << current << " м = " << kilometers
                    << " км; сума = " << cumulativeSum << '\n';
        current += step;
    }
}

// === ЗБЕРЕЖЕНО 3 ІНКРЕМЕНТУ 5, РЕФАКТОРИНГ ЧЕРЕЗ МАСИВИ ===
double recursiveConvert(double value, int currentUnit, int targetUnit)
{
    if (currentUnit == targetUnit)
    {
        return value;
    }

    const int nextUnit = currentUnit + 1;
    const double nextValue = convertLengthValue(value, currentUnit, nextUnit);
    return recursiveConvert(nextValue, nextUnit, targetUnit);
}

void printRecursiveChain(double value, int currentUnit, int targetUnit)
{
    if (currentUnit == targetUnit)
    {
        return;
    }

    const int nextUnit = currentUnit + 1;
    const double nextValue = convertLengthValue(value, currentUnit, nextUnit);
    std::cout << value << ' ' << UNIT_NAMES[currentUnit]
                << " = " << nextValue << ' ' << UNIT_NAMES[nextUnit] << '\n';
    printRecursiveChain(nextValue, nextUnit, targetUnit);
}

void runArbitraryUnitConversion(double value)
{
    printUnits();
}

```

```

    int sourceIndex = 0;
    int targetIndex = 0;
    if (!readUnitIndex("З якої одиниці: ", sourceIndex) ||
        !readUnitIndex("У яку одиницю: ", targetIndex))
    {
        return;
    }

    printResult(convertLengthValue(value, sourceIndex, targetIndex),
        UNIT_NAMES[targetIndex]);
}

void readValueArray(double values[], int count)
{
    for (int index = 0; index < count; ++index)
    {
        std::cout << "Значення [" << index << "]: ";
        while (!(std::cin >> values[index]))
        {
            std::cout << "Помилка. Повторіть введення [" << index << "]: ";
            clearInputStream();
        }
    }
}

void convertArray(const double sourceValues[],
    double convertedValues[],
    int count,
    double factor)
{
    for (int index = 0; index < count; ++index)
    {
        convertedValues[index] = sourceValues[index] * factor;
    }
}

void printArray(const double sourceValues[],
    const double convertedValues[],
    int count,
    const char sourceUnit[],
    const char targetUnit[])
{
    std::cout << std::setw(6) << "%"
        << std::setw(16) << sourceUnit
        << std::setw(16) << targetUnit << '\n';
    for (int index = 0; index < count; ++index)
    {
        std::cout << std::setw(6) << index
            << std::setw(16) << sourceValues[index]
            << std::setw(16) << convertedValues[index] << '\n';
    }
}

double findMinimum(const double values[], int count)
{
    double minimum = values[0];
    for (int index = 1; index < count; ++index)
    {
        if (values[index] < minimum)
        {
            minimum = values[index];
        }
    }
    return minimum;
}

```

```

double findMaximum(const double values[], int count)
{
    double maximum = values[0];
    for (int index = 1; index < count; ++index)
    {
        if (values[index] > maximum)
        {
            maximum = values[index];
        }
    }
    return maximum;
}

double calculateAverage(const double values[], int count)
{
    double sum = 0.0;
    for (int index = 0; index < count; ++index)
    {
        sum += values[index];
    }
    return sum / count;
}

int findNearestIndex(const double values[], int count, double target)
{
    int nearestIndex = 0;
    double nearestDistance = std::fabs(values[0] - target);

    for (int index = 1; index < count; ++index)
    {
        const double distance = std::fabs(values[index] - target);
        if (distance < nearestDistance)
        {
            nearestDistance = distance;
            nearestIndex = index;
        }
    }
    return nearestIndex;
}

void copyArray(const double source[], double destination[], int count)
{
    for (int index = 0; index < count; ++index)
    {
        destination[index] = source[index];
    }
}

void bubbleSort(double values[], int count)
{
    for (int pass = 0; pass < count - 1; ++pass)
    {
        bool swapped = false;
        for (int index = 0; index < count - 1 - pass; ++index)
        {
            if (values[index] > values[index + 1])
            {
                const double temporary = values[index];
                values[index] = values[index + 1];
                values[index + 1] = temporary;
                swapped = true;
            }
        }
        if (!swapped)

```

```

        {
            return;
        }
    }
}

int binarySearch(const double sortedValues[], int count, double target)
{
    int left = 0;
    int right = count - 1;

    while (left <= right)
    {
        const int middle = left + (right - left) / 2;
        if (std::fabs(sortedValues[middle] - target) < EPSILON)
        {
            return middle;
        }
        if (sortedValues[middle] < target)
        {
            left = middle + 1;
        }
        else
        {
            right = middle - 1;
        }
    }
    return -1;
}

void runArrayAnalysis()
{
    printUnits();
    int sourceIndex = 0;
    int targetIndex = 0;
    int count = 0;

    if (!readUnitIndex("Початкова одиниця: ", sourceIndex) ||
        !readUnitIndex("Кінцева одиниця: ", targetIndex) ||
        !readChoice("Кількість значень (1..20): ", 1, MAX_VALUE_COUNT, count))
    {
        return;
    }

    double sourceValues[MAX_VALUE_COUNT] = {};
    double convertedValues[MAX_VALUE_COUNT] = {};
    double sortedValues[MAX_VALUE_COUNT] = {};

    readValueArray(sourceValues, count);
    convertArray(sourceValues,
        convertedValues,
        count,
        conversionFactor(sourceIndex, targetIndex));
    printArray(sourceValues,
        convertedValues,
        count,
        UNIT_NAMES[sourceIndex],
        UNIT_NAMES[targetIndex]);

    std::cout << "Мінімум: " << findMinimum(convertedValues, count) << '\n';
    std::cout << "Максимум: " << findMaximum(convertedValues, count) << '\n';
    std::cout << "Середнє: " << calculateAverage(convertedValues, count) << '\n';

    double target = 0.0;
    if (readDouble("Значення для пошуку найближчого елемента: ", target))

```

```

    {
        const int nearest = findNearestIndex(convertedValues, count, target);
        std::cout << "Найближче значення: " << convertedValues[nearest]
            << ' ' << UNIT_NAMES[targetIndex] << '\n';
    }

    copyArray(convertedValues, sortedValues, count);
    bubbleSort(sortedValues, count);
    std::cout << "Відсортований масив: ";
    for (int index = 0; index < count; ++index)
    {
        std::cout << sortedValues[index] << ' ';
    }
    std::cout << '\n';

    if (readDouble("Значення для точного бінарного пошуку: ", target))
    {
        const int foundIndex = binarySearch(sortedValues, count, target);
        std::cout << (foundIndex >= 0 ? "Значення знайдено.\n"
            : "Значення не знайдено.\n");
    }
}

void convertLength(double value)
{
    std::cout << "1 - Метри -> кілометри\n";
    std::cout << "2 - Кілометри -> метри\n";
    std::cout << "3 - Рекурентна таблиця\n";
    std::cout << "4 - Рекурсивний ланцюжок мм -> км\n";
    std::cout << "5 - Конвертація через масив одиниць\n";
    std::cout << "6 - Обробка масиву значень\n";

    int choice = 0;
    if (!readChoice("Ваш вибір: ", 1, 6, choice))
    {
        return;
    }

    switch (choice)
    {
    case 1:
        printResult(value / 1000.0, "км");
        break;
    case 2:
        printResult(value * 1000.0, "м");
        break;
    case 3:
        printRecurrentTable();
        break;
    case 4:
        printRecursiveChain(value, 0, UNIT_COUNT - 1);
        printResult(recursiveConvert(value, 0, UNIT_COUNT - 1), "км");
        break;
    case 5:
        runArbitraryUnitConversion(value);
        break;
    case 6:
        runArrayAnalysis();
        break;
    }
}

void convertMass(double value)
{
    int direction = 0;

```

```

        std::cout << "1 - Грами -> кілограми\n2 - Кілограми -> грами\n";
        if (!readChoice("Ваш вибір: ", 1, 2, direction))
        {
            return;
        }
        printResult(direction == 1 ? value / 1000.0 : value * 1000.0,
            direction == 1 ? "кг" : "г");
    }

void convertTemperature(double value)
{
    int direction = 0;
    std::cout << "1 - Цельсій -> Фаренгейт\n2 - Фаренгейт -> Цельсій\n";
    if (!readChoice("Ваш вибір: ", 1, 2, direction))
    {
        return;
    }
    const double result = direction == 1
        ? value * CELSIUS_TO_FAHRENHEIT_SCALE + FAHRENHEIT_OFFSET
        : (value - FAHRENHEIT_OFFSET) / CELSIUS_TO_FAHRENHEIT_SCALE;
    printResult(result, direction == 1 ? "°F" : "°C");
}

void convertTime(double value)
{
    int direction = 0;
    std::cout << "1 - Секунди -> хвилини\n2 - Хвилини -> секунди\n";
    if (!readChoice("Ваш вибір: ", 1, 2, direction))
    {
        return;
    }
    printResult(direction == 1 ? value / SECONDS_PER_MINUTE
        : value * SECONDS_PER_MINUTE,
        direction == 1 ? "хв" : "с");
}

void runOneOperation()
{
    printHeader();
    std::cout << "1 - Довжина\n2 - Маса\n3 - Температура\n4 - Час\n";

    int category = 0;
    double value = 0.0;
    if (!readChoice("Категорія: ", 1, 4, category) ||
        !readDouble("Значення: ", value))
    {
        return;
    }

    switch (category)
    {
    case 1:
        convertLength(value);
        break;
    case 2:
        convertMass(value);
        break;
    case 3:
        convertTemperature(value);
        break;
    case 4:
        convertTime(value);
        break;
    }
}

```

```

int runApplication()
{
    do
    {
        runOneOperation();
        std::cout << '\n';
    } while (readRepeatAnswer());
    return 0;
}
} // namespace

int main()
{
    std::setlocale(LC_ALL, "");
    return runApplication();
}

```

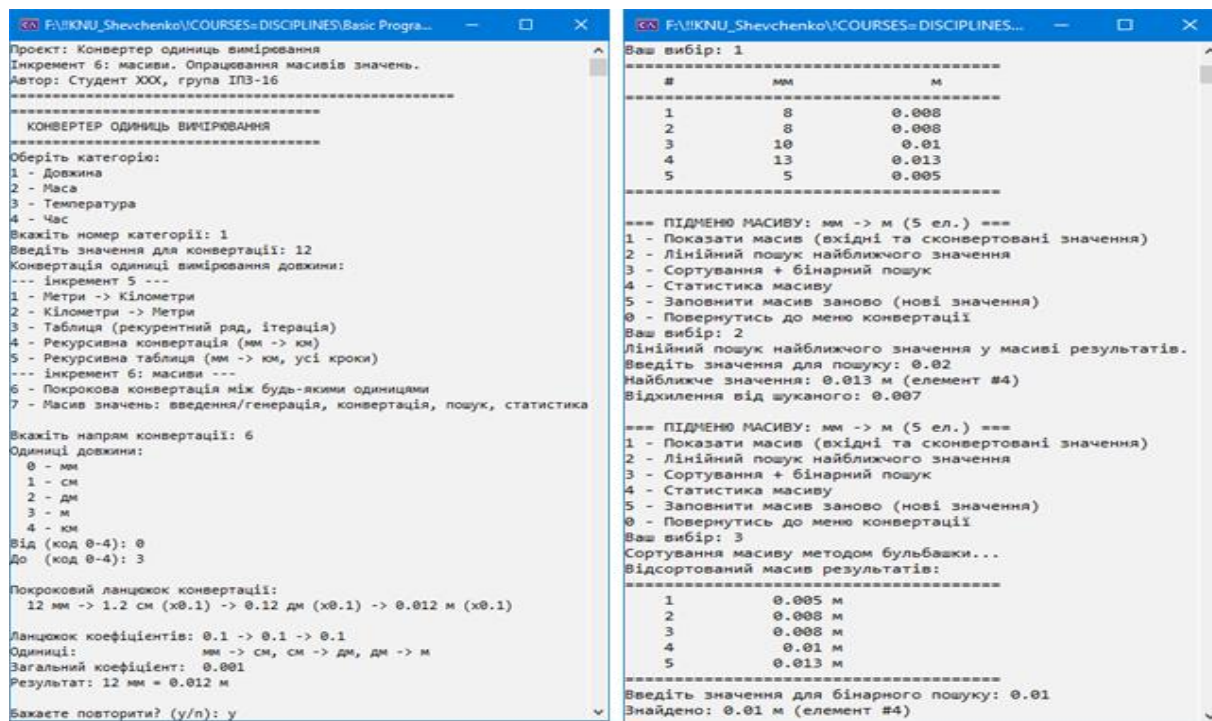


Рис. 4.12 – Фрагменти прикладу роботи інкременту 6

4.6.4 Контрольні запитання

1. Дати означення масиву та типу масиву. Якими є властивості масивів даних?
2. Яким є принцип зображення масиву в оперативній пам'яті?
3. Як здійснюється доступ до елементів одновимірного масиву?
4. Чи може список ініціалізації масиву містити більше (менше) значень, ніж вказано в оголошенні масиву?
5. Які базові операції обробки одновимірних масивів?
6. Чим відрізняються методи сортування масивів?
7. Приведіть найшвидший та найповільніший алгоритми сортування масивів.
8. Чим відрізняються алгоритми пошуку в упорядкованому та неупорядкованому масивах?
9. Чому при передачі одновимірного масиву у функцію фактично передається не сам масив, а інша сутність? Які наслідки це має для можливості зміни даних у функції?

10. Чому в мові C++ не виконується автоматична перевірка виходу за межі масиву, і які потенційні ризики це створює для програм?

4.7 Лабораторна робота №7. Багатовимірні масиви

4.7.1 Мета та завдання

Мета роботи:

1. Ознайомитися з поняттям багатовимірних масивів та їх представленням у мові C++.
2. Вивчити способи оголошення, ініціалізації та використання двовимірних і багатовимірних масивів.
3. Набути практичних навичок обробки табличних даних.
4. Засвоїти методи роботи з багатовимірними структурами за допомогою вкладених циклів.

Завдання роботи:

1. Вивчити синтаксис оголошення та ініціалізації багатовимірних масивів.
2. Ознайомитися зі способами введення та виведення елементів масиву у вигляді таблиць.
3. Розробити програму для обробки двовимірного масиву (матриці).
4. Реалізувати обхід елементів масиву за допомогою вкладених циклів.
5. Виконати типові операції над матрицями (обчислення суми елементів, пошук максимального/мінімального значення, обробка рядків і стовпців).
6. Провести тестування програми та проаналізувати результати її роботи.

4.7.2 Теоретичні відомості

Синтаксис оголошення змінної для подання матриці в мовах C/C++:

<тип елементів> <ім'я матриці> [<кількість рядків>][<кількість стовпців>;

У цьому оголошенні <тип елементів> - будь-який тип даних, окрім файлового типу; <ім'я матриці> — деякий ідентифікатор; <кількість рядків> і <кількість стовпців> — константи, що визначає розмірність матриці. Межі діапазону допустимих значень індексів рядків і стовпців визначаються значеннями від 0 до <кількість рядків>-1 та від 0 до <кількість стовпців>-1. Масиви, що мають більш ніж два виміри, оголошуються в аналогічний спосіб.

Доступ до елементів матриці здійснюється операцією індексування [] за двома індексами, які визначають номер рядка та номер стовпця елемента. Синтаксис операції індексування є таким:

<ім'я масиву> [<номер рядка>][<номер стовпця>]

Базові операції обробки двовимірних масивів

Перелік базових операцій над матрицями та їх елементами:

- створення нової матриці за заданим алгоритмом;
- введення та виведення матриць;
- пошук елементів матриці за певним критерієм;
- визначення, чи задовольняє матриця або окремі її елементи певній властивості;

- виконання певних операцій над компонентами матриць (переставлення рядків і стовпців, множення матриць тощо).

Введення матриці:

```
int a[5][5];
int main()
{
    for(int i = 0; i < 5; i++)           //зовнішній цикл по рядках
        for(int j = 0; j < 5; j++)       //внутрішній цикл по стовпцях
            cin >> a[i][j];             //із клавіатури елементи
}
```

Під час виведення матриці переводити курсор на новий рядок слід лише після виведення всіх елементів поточного рядка:

```
for(int i = 0; i < 5; i++)
{
    for(int j = 0; j < 5; j++)
        cout << a[i][j] << " ";       //вивести елементи рядка
    cout << endl;                       //перевести курсор на новий рядок
}
```

Ініціалізація елементів матриці деякими значеннями:

```
int a[5][5];
int main()
{
    for(int i = 0; i < 5; i++)
        for(int j = 0; j < 5; j++)
            a[i][j] = 10; // альтернативи: a[i][j] = rand() % 20;
                           // a[i][j] = i * j; тощо
}
```

Пошук значення та індексів максимального елемента матриці:

```
max = a[0][0]; //початкове значення максимального елемента
for( int i = 0; i < n; i++)
    for( int j = 0; j < n; j++)
        if ( max < a[i][j])
        {
            max = a[i][j]; //запам'ятати максимальний елемент
            imax = i;       //та його індекси
            jmax = j;
        }
```

Перестановка рядків або стовпців

```
for(i=0;i<n;i++) //цикл по рядках
{
    tmp = a[i][0]; //альтернатива для цілих значень:
    a[i][0] = a[i][4]; //a[i][4] ^= a[i][0] ^= a[i][4] ^= a[i][0];
    a[i][4] = tmp;
}
```

4.7.3 Приклад виконання

Розглянемо приклад виконання сьомого інкременту(лабораторної роботи).

7	Багатовимірні масиви	Матриця коефіцієнтів конвертації між одиницями. Табличне перетворення значень	Таблична система конвертації
---	----------------------	---	------------------------------

Вимоги до написання коду

У цьому інкременті необхідно розширити програму засобами роботи з двовимірними масивами. Двовимірний масив слід використати для зберігання коефіцієнтів конвертації між одиницями вимірювання або результатів табличного перетворення.

Під час написання програми необхідно:

1. Оголосити двовимірний масив відповідного типу для зберігання коефіцієнтів або результатів конвертації.
2. Розміри масиву задавати іменованими константами, наприклад:
`const int ROWS = 4;`
`const int COLS = 4;`
3. Не використовувати змінні значення як розміри статичного масиву.
4. Забезпечити, щоб індекси рядків і стовпців належали допустимим діапазорам:
`0 <= row && row < ROWS`
`0 <= column && column < COLS`
5. Для введення, заповнення, обробки та виведення двовимірного масиву застосовувати вкладені цикли.
6. Узгодити призначення індексів: перший індекс повинен визначати рядок або початкову одиницю, другий — стовпець або кінцеву одиницю.
7. Використовувати змістовні назви масивів і змінних, наприклад:
`conversionRates`
`conversionTable`
`sourceUnit`
`targetUnit`
8. Матрицю коефіцієнтів, яку програма не повинна змінювати, передавати у функції з модифікатором `const`.
9. Винести основні операції над матрицею в окремі функції. Доцільно передбачити функції для:
 - заповнення матриці коефіцієнтів;
 - виведення матриці;
 - перевірки індексів вибраних одиниць;
 - обчислення результату конвертації;
 - формування таблиці результатів.
10. Під час передавання статичного двовимірного масиву у функцію обов'язково зазначити кількість стовпців.
Приклад заголовка функції:
`void printMatrix(
 const double matrix[][COLS],
 int rows
);`
11. Не допускати звернення до елементів за межами масиву.
12. У функції `main()` залишити організацію взаємодії з користувачем і послідовність викликів функцій, а обробку матриці винести в окремі функції.

13. Перевіряти правильність номерів початкової та кінцевої одиниць вимірювання.
14. Виводити матрицю або таблицю результатів у зрозумілому вигляді із заголовками рядків і стовпців.
15. Дотримуватися єдиного стилю оформлення коду, використовувати відступи та короткі коментарі до основних логічних блоків.

Результатом виконання інкременту має бути програма, яка зберігає коефіцієнти конвертації у двовимірному масиві, отримує від користувача початкову й кінцеву одиниці та виконує табличне перетворення введеного значення.

Рис. 4.13 демонструє HIPO діаграму інкременту 7.

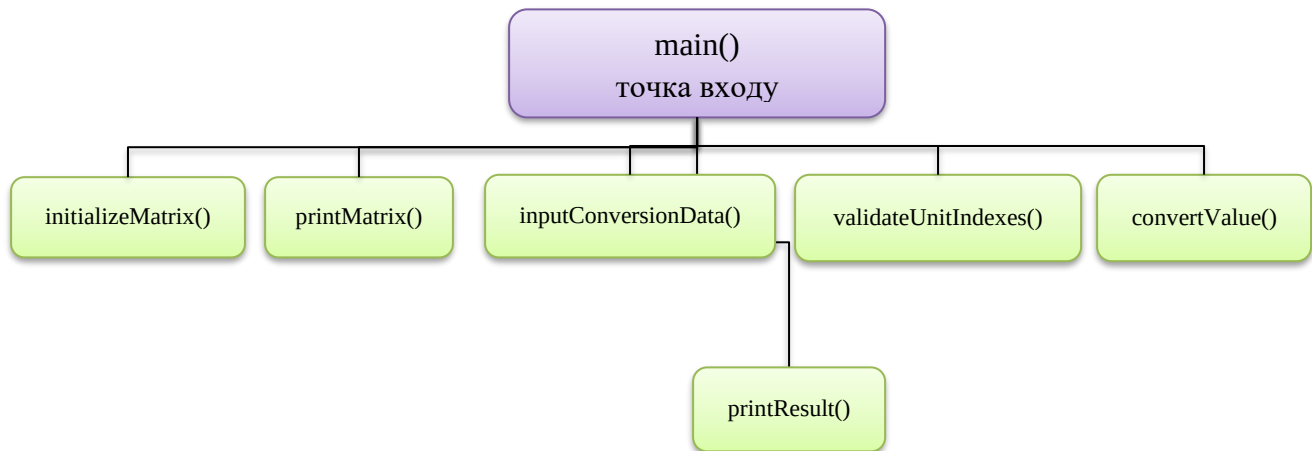


Рис. 4.13 – HIPO діаграма інкременту 7

Приклад коду та результату роботи

```

/*
* Проект:    Програмна система конвертації одиниць вимірювання
* Інкремент: 7 – багатовимірні масиви
*
* ДОДАНО В ЦЬОМУ ІНКРЕМЕНТІ:
* - двовимірну матрицю коефіцієнтів конвертації;
* - формування матриці з масиву коефіцієнтів до базової одиниці;
* - табличне виведення матриці та конвертацію за двома індексами;
* - перевірку індексів перед зверненням до елемента matrix[row][column].
*
* РЕФАКТОРИНГ ПОПЕРЕДНЬОГО ІНКРЕМЕНТУ:
* - збережено функції роботи з одновимірними масивами;
* - матриця не дублює вручну введені «магічні» коефіцієнти, а будується
*   з єдиного джерела TO_MILLIMETERS (DRY);
* - робота з матрицею винесена в окремі функції (SRP).
*
* НАВЧАЛЬНЕ СПРОЩЕННЯ:
* Розмір матриці фіксований на етапі компіляції. Це полегшує вивчення
* індексації та вкладених циклів, але не дає додавати одиниці під час роботи.
*
* ПЛАНОВИЙ РЕФАКТОРИНГ:
* В інкременті 8 матриця й масив назв стануть динамічними; буде додано
* ручне керування пам'яттю, покажчики, посилання та покажчики на функції.
*/

```

```

#include <locale>
#include <cmath>
#include <iomanip>
#include <iostream>
#include <limits>

namespace
{
    constexpr int UNIT_COUNT = 5;
    constexpr int MAX_VALUE_COUNT = 20;
    constexpr double EPSILON = 1e-9;
    constexpr double SECONDS_PER_MINUTE = 60.0;
    constexpr double CELSIUS_TO_FAHRENHEIT_SCALE = 9.0 / 5.0;
    constexpr double FAHRENHEIT_OFFSET = 32.0;

    const char* const UNIT_NAMES[UNIT_COUNT] = { "мм", "см", "дм", "м", "км" };
    const double TO_MILLIMETERS[UNIT_COUNT] = { 1.0, 10.0, 100.0, 1000.0, 1000000.0
};

    void clearInputStream()
    {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }

    bool readInt(const char prompt[], int& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }
        std::cout << "Помилка: потрібно ввести ціле число.\n";
        clearInputStream();
        return false;
    }

    bool readDouble(const char prompt[], double& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }
        std::cout << "Помилка: потрібно ввести число.\n";
        clearInputStream();
        return false;
    }

    bool readChoice(const char prompt[], int minimum, int maximum, int& value)
    {
        if (!readInt(prompt, value))
        {

```

```

        return false;
    }
    if (value < minimum || value > maximum)
    {
        std::cout << "Помилка: допустимий діапазон "
            << minimum << ".." << maximum << "...\n";
        return false;
    }
    return true;
}

bool readRepeatAnswer()
{
    char answer = 'n';
    std::cout << "Бажаєте виконати ще одну операцію? (y/n): ";
    if (!(std::cin >> answer))
    {
        clearInputStream();
        return false;
    }
    return answer == 'y' || answer == 'Y';
}

void printHeader()
{
    std::cout << "=====\n";
    std::cout << "  КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ\n";
    std::cout << "=====\n";
}

void printResult(double result, const char unit[])
{
    std::cout << "Результат: " << result << ' ' << unit << '\n';
}

void printUnits()
{
    std::cout << "Одиниці довжини:\n";
    for (int index = 0; index < UNIT_COUNT; ++index)
    {
        std::cout << index << " - " << UNIT_NAMES[index] << '\n';
    }
}

bool readUnitIndex(const char prompt[], int& index)
{
    return readChoice(prompt, 0, UNIT_COUNT - 1, index);
}

double conversionFactor(int sourceIndex, int targetIndex)
{
    return TO_MILLIMETERS[sourceIndex] / TO_MILLIMETERS[targetIndex];
}

```

```

double convertLengthValue(double value, int sourceIndex, int targetIndex)
{
    return value * conversionFactor(sourceIndex, targetIndex);
}

void printRecurrentTable()
{
    double start = 0.0;
    double finish = 0.0;
    double step = 0.0;

    if (!readDouble("Початкове значення, м: ", start) ||
        !readDouble("Кінцеве значення, м: ", finish) ||
        !readDouble("Крок: ", step))
    {
        return;
    }
    if (finish < start || step <= 0.0)
    {
        std::cout << "Помилка: перевірте межі та крок.\n";
        return;
    }

    double current = start;
    double cumulativeSum = 0.0;
    while (current <= finish)
    {
        const double kilometers = current / 1000.0;
        cumulativeSum += kilometers;
        std::cout << current << " м = " << kilometers
                    << " км; сума = " << cumulativeSum << '\n';
        current += step;
    }
}

// === ЗБЕРЕЖЕНО 3 ІНКРЕМЕНТУ 5, РЕФАКТОРИНГ ЧЕРЕЗ МАСИВИ ===
double recursiveConvert(double value, int currentUnit, int targetUnit)
{
    if (currentUnit == targetUnit)
    {
        return value;
    }

    const int nextUnit = currentUnit + 1;
    const double nextValue = convertLengthValue(value, currentUnit, nextUnit);
    return recursiveConvert(nextValue, nextUnit, targetUnit);
}

void printRecursiveChain(double value, int currentUnit, int targetUnit)
{
    if (currentUnit == targetUnit)
    {

```

```

        return;
    }

    const int nextUnit = currentUnit + 1;
    const double nextValue = convertLengthValue(value, currentUnit, nextUnit);
    std::cout << value << ' ' << UNIT_NAMES[currentUnit]
        << " = " << nextValue << ' ' << UNIT_NAMES[nextUnit] << '\n';
    printRecursiveChain(nextValue, nextUnit, targetUnit);
}

void runArbitraryUnitConversion(double value)
{
    printUnits();
    int sourceIndex = 0;
    int targetIndex = 0;
    if (!readUnitIndex("З якої одиниці: ", sourceIndex) ||
        !readUnitIndex("У яку одиницю: ", targetIndex))
    {
        return;
    }

    printResult(convertLengthValue(value, sourceIndex, targetIndex),
        UNIT_NAMES[targetIndex]);
}

void readValueArray(double values[], int count)
{
    for (int index = 0; index < count; ++index)
    {
        std::cout << "Значення [" << index << "]: ";
        while (!(std::cin >> values[index]))
        {
            std::cout << "Помилка. Повторіть введення [" << index << "]: ";
            clearInputStream();
        }
    }
}

void convertArray(const double sourceValues[],
    double convertedValues[],
    int count,
    double factor)
{
    for (int index = 0; index < count; ++index)
    {
        convertedValues[index] = sourceValues[index] * factor;
    }
}

void printArray(const double sourceValues[],
    const double convertedValues[],
    int count,
    const char sourceUnit[],

```

```

    const char targetUnit[])
{
    std::cout << std::setw(6) << "%s"
        << std::setw(16) << sourceUnit
        << std::setw(16) << targetUnit << '\n';
    for (int index = 0; index < count; ++index)
    {
        std::cout << std::setw(6) << index
            << std::setw(16) << sourceValues[index]
            << std::setw(16) << convertedValues[index] << '\n';
    }
}

double findMinimum(const double values[], int count)
{
    double minimum = values[0];
    for (int index = 1; index < count; ++index)
    {
        if (values[index] < minimum)
        {
            minimum = values[index];
        }
    }
    return minimum;
}

double findMaximum(const double values[], int count)
{
    double maximum = values[0];
    for (int index = 1; index < count; ++index)
    {
        if (values[index] > maximum)
        {
            maximum = values[index];
        }
    }
    return maximum;
}

double calculateAverage(const double values[], int count)
{
    double sum = 0.0;
    for (int index = 0; index < count; ++index)
    {
        sum += values[index];
    }
    return sum / count;
}

int findNearestIndex(const double values[], int count, double target)
{
    int nearestIndex = 0;
    double nearestDistance = std::fabs(values[0] - target);

```

```

    for (int index = 1; index < count; ++index)
    {
        const double distance = std::fabs(values[index] - target);
        if (distance < nearestDistance)
        {
            nearestDistance = distance;
            nearestIndex = index;
        }
    }
    return nearestIndex;
}

void copyArray(const double source[], double destination[], int count)
{
    for (int index = 0; index < count; ++index)
    {
        destination[index] = source[index];
    }
}

void bubbleSort(double values[], int count)
{
    for (int pass = 0; pass < count - 1; ++pass)
    {
        bool swapped = false;
        for (int index = 0; index < count - 1 - pass; ++index)
        {
            if (values[index] > values[index + 1])
            {
                const double temporary = values[index];
                values[index] = values[index + 1];
                values[index + 1] = temporary;
                swapped = true;
            }
        }
        if (!swapped)
        {
            return;
        }
    }
}

int binarySearch(const double sortedValues[], int count, double target)
{
    int left = 0;
    int right = count - 1;

    while (left <= right)
    {
        const int middle = left + (right - left) / 2;
        if (std::fabs(sortedValues[middle] - target) < EPSILON)
        {

```

```

        return middle;
    }
    if (sortedValues[middle] < target)
    {
        left = middle + 1;
    }
    else
    {
        right = middle - 1;
    }
}
return -1;
}

void runArrayAnalysis()
{
    printUnits();
    int sourceIndex = 0;
    int targetIndex = 0;
    int count = 0;

    if (!readUnitIndex("Початкова одиниця: ", sourceIndex) ||
        !readUnitIndex("Кінцева одиниця: ", targetIndex) ||
        !readChoice("Кількість значень (1..20): ", 1, MAX_VALUE_COUNT, count))
    {
        return;
    }

    double sourceValues[MAX_VALUE_COUNT] = {};
    double convertedValues[MAX_VALUE_COUNT] = {};
    double sortedValues[MAX_VALUE_COUNT] = {};

    readValueArray(sourceValues, count);
    convertArray(sourceValues,
        convertedValues,
        count,
        conversionFactor(sourceIndex, targetIndex));
    printArray(sourceValues,
        convertedValues,
        count,
        UNIT_NAMES[sourceIndex],
        UNIT_NAMES[targetIndex]);

    std::cout << "Мінімум: " << findMinimum(convertedValues, count) << '\n';
    std::cout << "Максимум: " << findMaximum(convertedValues, count) << '\n';
    std::cout << "Середнє: " << calculateAverage(convertedValues, count) << '\n';

    double target = 0.0;
    if (readDouble("Значення для пошуку найближчого елемента: ", target))
    {
        const int nearest = findNearestIndex(convertedValues, count, target);
        std::cout << "Найближче значення: " << convertedValues[nearest]
            << ' ' << UNIT_NAMES[targetIndex] << '\n';
    }
}

```

```

    }

    copyArray(convertedValues, sortedValues, count);
    bubbleSort(sortedValues, count);
    std::cout << "Відсортований масив: ";
    for (int index = 0; index < count; ++index)
    {
        std::cout << sortedValues[index] << ' ';
    }
    std::cout << '\n';

    if (readDouble("Значення для точного бінарного пошуку: ", target))
    {
        const int foundIndex = binarySearch(sortedValues, count, target);
        std::cout << (foundIndex >= 0 ? "Значення знайдено.\n"
            : "Значення не знайдено.\n");
    }
}

// === ДОДАНО В ІНКРЕМЕНТІ 7: ДВОВИМІРНИЙ МАСИВ ===
void buildConversionMatrix(double matrix[][UNIT_COUNT])
{
    for (int source = 0; source < UNIT_COUNT; ++source)
    {
        for (int target = 0; target < UNIT_COUNT; ++target)
        {
            matrix[source][target] = conversionFactor(source, target);
        }
    }
}

void printConversionMatrix(const double matrix[][UNIT_COUNT])
{
    std::cout << std::setw(8) << "";
    for (int target = 0; target < UNIT_COUNT; ++target)
    {
        std::cout << std::setw(12) << UNIT_NAMES[target];
    }
    std::cout << '\n';

    for (int source = 0; source < UNIT_COUNT; ++source)
    {
        std::cout << std::setw(8) << UNIT_NAMES[source];
        for (int target = 0; target < UNIT_COUNT; ++target)
        {
            std::cout << std::setw(12) << matrix[source][target];
        }
        std::cout << '\n';
    }
}

void runMatrixConversion(double value)

```

```

{
    double conversionMatrix[UNIT_COUNT][UNIT_COUNT] = {};
    buildConversionMatrix(conversionMatrix);
    printConversionMatrix(conversionMatrix);

    int sourceIndex = 0;
    int targetIndex = 0;
    if (!readUnitIndex("Рядок – початкова одиниця: ", sourceIndex) ||
        !readUnitIndex("Стовпець – кінцева одиниця: ", targetIndex))
    {
        return;
    }

    const double result = value * conversionMatrix[sourceIndex][targetIndex];
    printResult(result, UNIT_NAMES[targetIndex]);
}

void convertLength(double value)
{
    std::cout << "1 - Метри -> кілометри\n";
    std::cout << "2 - Кілометри -> метри\n";
    std::cout << "3 - Рекурентна таблиця\n";
    std::cout << "4 - Рекурсивний ланцюжок мм -> км\n";
    std::cout << "5 - Конвертація через масив одиниць\n";
    std::cout << "6 - Обробка масиву значень\n";
    std::cout << "7 - Конвертація через матрицю коефіцієнтів\n";

    int choice = 0;
    if (!readChoice("Ваш вибір: ", 1, 7, choice))
    {
        return;
    }

    switch (choice)
    {
    case 1:
        printResult(value / 1000.0, "км");
        break;
    case 2:
        printResult(value * 1000.0, "м");
        break;
    case 3:
        printRecurrentTable();
        break;
    case 4:
        printRecursiveChain(value, 0, UNIT_COUNT - 1);
        printResult(recursiveConvert(value, 0, UNIT_COUNT - 1), "км");
        break;
    case 5:
        runArbitraryUnitConversion(value);
        break;
    case 6:
        runArrayAnalysis();

```

```

        break;
    case 7:
        runMatrixConversion(value);
        break;
    }
}

void convertMass(double value)
{
    int direction = 0;
    std::cout << "1 - Грами -> кілограми\n2 - Кілограми -> грами\n";
    if (!readChoice("Ваш вибір: ", 1, 2, direction))
    {
        return;
    }
    printResult(direction == 1 ? value / 1000.0 : value * 1000.0,
        direction == 1 ? "кг" : "г");
}

void convertTemperature(double value)
{
    int direction = 0;
    std::cout << "1 - Цельсій -> Фаренгейт\n2 - Фаренгейт -> Цельсій\n";
    if (!readChoice("Ваш вибір: ", 1, 2, direction))
    {
        return;
    }
    const double result = direction == 1
        ? value * CELSIUS_TO_FAHRENHEIT_SCALE + FAHRENHEIT_OFFSET
        : (value - FAHRENHEIT_OFFSET) / CELSIUS_TO_FAHRENHEIT_SCALE;
    printResult(result, direction == 1 ? "°F" : "°C");
}

void convertTime(double value)
{
    int direction = 0;
    std::cout << "1 - Секунди -> хвилини\n2 - Хвилини -> секунди\n";
    if (!readChoice("Ваш вибір: ", 1, 2, direction))
    {
        return;
    }
    printResult(direction == 1 ? value / SECONDS_PER_MINUTE
        : value * SECONDS_PER_MINUTE,
        direction == 1 ? "хв" : "с");
}

void runOneOperation()
{
    printHeader();
    std::cout << "1 - Довжина\n2 - Маса\n3 - Температура\n4 - Час\n";

    int category = 0;
    double value = 0.0;

```

```

        if (!readChoice("Категорія: ", 1, 4, category) ||
            !readDouble("Значення: ", value))
        {
            return;
        }

        switch (category)
        {
        case 1:
            convertLength(value);
            break;
        case 2:
            convertMass(value);
            break;
        case 3:
            convertTemperature(value);
            break;
        case 4:
            convertTime(value);
            break;
        }
    }

    int runApplication()
    {
        do
        {
            runOneOperation();
            std::cout << '\n';
        } while (readRepeatAnswer());
        return 0;
    }
} // namespace

int main()
{
    std::setlocale(LC_ALL, "");
    return runApplication();
}

```

```
КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ
=====
Введіть значення: 563

Оберіть категорію:
1 - Довжина
2 - Маса
3 - Температура
4 - Час
Ваш вибір: 1

Конвертація довжини:
1 - Метри -> Кілометри
2 - Кілометри -> Метри
3 - Таблиця (1-10 метрів у кілометри)
4 - Багатокроково (мм -> см -> м -> км)
5 - Рекурсивна конвертація (мм -> км)
6 - Конвертація через масив одиниць
7 - Таблична конвертація (матриця)
Ваш вибір: 7

Одиниць?:
0 - мм
1 - см
2 - м
3 - км
З якої одиниць?: 3
В яку одиницю: 0
Результат: 5.63e+08 мм

Бажаєте продовжити? (y/n):
```

Рис. 4.14 Приклад роботи інкременту 7

4.7.4 Контрольні запитання

1. Дати означення матриці.
2. Як зображується матриця в оперативній пам'яті?
3. Як здійснити доступ до елементів масиву чи матриці?
4. Як здійснити доступ до елемента матриці, користуючись значенням тільки одного індексу?
5. Означити типові операції над матрицями.
6. Чи існують у пам'яті комп'ютера «рядки» та «стовпці»? Як комп'ютер обчислює адресу конкретного елемента матриці?
7. Чому не можна написати `int n; cin >> n; int A[n][n];` у стандартному C++?
8. Що станеться, якщо до матриці `int A[3][3]` звернутися за індексом `A[0][5]`? Чи видасть компілятор помилку?
9. Є два цикли для обчислення суми елементів матриці: один проходить спочатку по рядках, а інший — по стовпцях. Який працюватиме швидше і чому?
10. Чому при передачі двовимірного статичного масиву у функцію, наприклад, `void func(int arr[][10])`, ми маємо право не вказувати першу розмірність, але **зобов'язані** чітко вказати другу (кількість стовпців)? Що буде, якщо ми її пропустимо або вкажемо неправильно?

4.8 Лабораторна робота №8. Показчики та посилання. Показчики на функції. Масиви показчиків

4.8.1 Мета та завдання

Мета роботи:

1. Ознайомитися з поняттям показчиків та посилань у мові C++.
2. Вивчити принципи роботи з адресами пам'яті та непрямим доступом до даних.
3. Набути практичних навичок використання показчиків, посилань і показчиків на функції.
4. Засвоїти особливості роботи з масивами показчиків та їх застосування.

Завдання роботи:

1. Вивчити синтаксис оголошення та ініціалізації показчиків і посилань.
2. Ознайомитися з операціями розіменування (*) та взяття адреси (&).
3. Розробити програму, що демонструє використання показчиків для доступу та зміни значень змінних.
4. Реалізувати передачу параметрів у функції за допомогою показчиків та посилань.
5. Ознайомитися з поняттям показчиків на функції та реалізувати їх використання у програмі.
6. Створити та використати масив показчиків для обробки даних або виклику функцій.
7. Провести тестування програми та проаналізувати результати її роботи.

4.8.2 Теоретичні відомості

До посилальних типів відносяться показчики та посилання. Показчики дозволяють працювати з адресами комірок оперативної пам'яті, тим самим вони реалізують непрямий доступ до їх вмісту. Посилання є альтернативними іменами змінних. Застосування показчиків і посилань як параметрів функцій дозволяє функції змінювати значення своїх аргументів, отже, повертати в програму більше ніж одне значення.

Змінні, що зберігають адреси інших змінних, називаються *змінними-показчиками* чи просто *показчиками*.

Для визначення адреси змінної означена операція & отримання адреси.

Якщо оголошена та ініціалізована змінна, наприклад, `int value=10`, її адресу можна визначити виразом `&value`.

Якщо значення, що зберігаються, займатимуть послідовність байтів, показчик адресує перший байт цієї послідовності.

Оголошення та ініціалізація

Константні значення, арифметичні вирази та регістрові змінні не зберігають значення в оперативній пам'яті, тому застосувати до них операцію & неприпустимо.

Змінну-показчик оголошують:

`<тип>* <ідентифікатор_показчика>;`

Тут `<тип>` – простий чи структурований тип адресованої змінної; `<ідентифікатор_показчика>` - рядок символів, що є ім'ям змінної-показчика; символ `*` означає «вказати на».

Показчик перед використанням ініціалізується адресою змінної або значенням іншого показника. Показчику можна присвоїти значення 0 (NULL). Такий показчик не адресує жодну змінну.

Посилання є псевдонімом змінної, тобто її альтернативним іменем і для нього не резервується місце в оперативній пам'яті. Синтаксис оголошення посилання застосовує символ &, який записують після типу змінної:

<тип>& <ідентифікатор_посилання >;

Посилання слід ініціалізувати ім'ям змінної. Типи посилання та змінної, значенням якої ініціалізуються посилання, мають збігатися.

Операції над показниками

Для показників означені операції *адресування, присвоєння, арифметичні та відношення*.

1. Унарна операція & повертає адресу свого аргументу.
2. Значення змінної за певною адресою отримують, застосувавши операцію *розіменування*. Операція позначається символом *, який записують перед іменем показника. Операція * повертає об'єкт, на який вказує показчик.
3. Операція *присвоєння* використовується для надання значення показчику.
4. Арифметичні операції:
 - a. унарні операції інкремента та декремента,
 - b. бінарні операції додавання та віднімання цілого числа,
 - c. віднімання одного показника з іншого.
5. Порівняння показників. Показники одного типу можна *порівнювати* один з одним.

При цьому порівнюються адреси, що зберігаються в показниках.

Показчик на функцію містить адресу її в оперативній пам'яті. Ім'я функції – це початкова адреса її коду. До функцій можна застосувати тільки дві операції: виклик і отримання її адреси, тобто визначити її показчик.

Синтаксис оголошення показника на функцію:

<тип> (<ідентифікатор показника>)(<оголошення параметрів>);*

Щоб викликати функцію через показчик, його слід розіменувати.

Існують три способи передачі аргументів у функцію - *як значення, як посилання та як показники*.

Щоб повертати з функцій більше одного значення, слід передати функції аргументи-посилання або аргументи-показники. Під час оголошення функції параметри-посилання вказують у її заголовку, використовуючи операцію посилання &. Параметри-показники оголошують в заголовку функції, використовуючи операцію непрямої адресації *.

В операціях виклику функції з аргументом-показником застосовується операція & адресації змінної, значення котрої змінюватимуться. В операції виклику функції з параметром-посилання аргументом є ім'я змінної.

Обробка розріджених матриць

Розрідженою матрицею називається матриця, в якій кількість нульових елементів набагато переважає над ненульовими. У таких матрицях кількість ненульових елементів становить до 20%. Робота з розріджених матрицею передбачає особливі способи доступу до елементів. Для спрощення цієї роботи були розроблені спеціальні формати зберігання ненульових елементів матриці.

Формати зберігання розрідженої матриці. Існує п'ять найбільш поширених форматів зберігання розріджених матриць:

1. Coordinate Storage (COO - координатний формат),
2. Compressed Row Storage (CRS - розріджений рядковий формат),
3. Compressed Column Storage (CCS - розріджений стовпцовий формат),
4. Row-wise Representation Complete and Ordered (RR (C) O - упорядкований рядковий формат),
5. Column-wise Representation Complete and Ordered (CR (C) O - упорядкований стовпцовий формат).

Координатний формат COO. Матриця представляється у вигляді трьох одновимірних масивів, перший з яких зберігає значення ненульових елементів матриці, другий і третій - номери рядків і стовпців цих елементів відповідно. Розміри масивів відповідають кількості ненульових елементів. Приклад збереження розрідженої матриці за форматом COO:

2					
3					
		4			
1			5		
			3	7	
					8
	1			2	6

2	3	4	1	5	3	7	8	1	2	6
---	---	---	---	---	---	---	---	---	---	---

VALUE

0	1	2	3	3	4	4	5	6	6	6
---	---	---	---	---	---	---	---	---	---	---

ROW

0	1	2	0	3	3	4	5	1	5	6
---	---	---	---	---	---	---	---	---	---	---

COL

Розріджений рядковий формат CRS. Матриця A представляється у вигляді трьох одновимірних масивів. Значення ненульових елементів зберігаються в масиві AN , відповідні їм індекси стовпців - в масиві JA . Розмірності NNZ цих одновимірних масивів однакові, дорівнюють кількості ненульових елементів розрідженої матриці. Крім того, використовується масив показчиків рядків IA , які відзначають позиції ненульового елемента AN і відповідного стовпця JA , з яких починаються опис чергового рядка. Тобто на k -ому місці в масиві IA зберігається позиція першого ненульового елемента k -ого рядка в масиві AN . Масив показчиків може містити додатковий кінцевий елемент, для якого встановлено значення NNZ . Приклад збереження розрідженої матриці за форматом CRS:

Значення матриці

Індекси стовпців

Масив показчиків на рядки

Упорядкований рядковий формат RR (C) O. Матриця A представляється у вигляді трьох одновимірних масивів. Значення ненульових елементів зберігаються в масиві AN , відповідні їм індекси стовпців - в масиві JA . Розмірності NNZ цих одновимірних масивів однакові, дорівнюють кількості ненульових елементів розрідженої матриці. Масив показчиків IA використовується для посилання на компоненти масивів AN і JA , з яких починається опис чергового рядка. Остання компонента масиву IA містить показник першої вільної компоненти в масивах AN і JA , тобто дорівнює кількості ненульових елементів матриці, збільшеній на одиницю. Приклад збереження розрідженої матриці за форматом $RR (C) O$:

	1	2	3	4	5	6	7	8	9	10
1	0	3	0	1	0	0	0	7	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	8	0	9	0	0

AN	3	1	7	8	9
JA	2	4	8	6	8
IA	1	4	4	6	

Масив *IA* містить номер, з якого починається опис елементів в масивах *AN* і *JA*. Таким чином, інформація про елементи 3-го рядка матриці зберігатися в елементах з $IA[3] = 4$ по $IA[4] - 1 = 5$ масивів *AN* і *JA*. Оскільки $IA(2) = IA(3) = 4$, то це означає, що другий рядок матриці *A* нульовий

4.8.3 Приклад виконання

Розглянемо приклад виконання восьмого інкременту(лабораторної роботи).

8	Динамічна пам'ять і показники	Динамічне додавання нових одиниць вимірювання. Розширення системи конвертації	Гнучка система підтримки одиниць
---	-------------------------------	--	----------------------------------

Вимоги до написання коду

У восьмому інкременті необхідно забезпечити динамічне зберігання списку одиниць вимірювання та матриці коефіцієнтів конвертації. Користувач повинен мати можливість додати нову одиницю, після чого розмір масивів збільшується.

Під час написання програми необхідно:

1. Використати показник типу `string*` для динамічного масиву назв одиниць.
2. Використати показник типу `double**` для динамічної матриці коефіцієнтів.
3. Виділяти пам'ять для масивів за допомогою оператора `new[]`.
4. Перевіряти, щоб показники були ініціалізовані до першого використання.
5. Після створення нових масивів скопіювати до них усі дані з попередніх масивів.
6. Додати назву нової одиниці до останнього елемента розширеного масиву.
7. Заповнити новий рядок і новий стовпець матриці коректними коефіцієнтами конвертації.
8. Звільнити пам'ять, виділену для старих масивів, лише після успішного копіювання даних.
9. Для кожного рядка динамічної матриці окремо виконати `delete[]`, після чого звільнити масив показників.
10. Після заміни масивів оновити показники `units`, `matrix` і значення `size`.
11. Не допускати витоку пам'яті, повторного звільнення пам'яті та звернення до вже звільнених ділянок.
12. Перевіряти допустимість індексів початкової та кінцевої одиниць перед зверненням до матриці.
13. Основні операції з динамічною пам'яттю доцільно винести в окремі функції: створення масивів, додавання одиниці, копіювання даних і звільнення пам'яті.

14. Для передавання покажчиків, які мають бути змінені всередині функції, використовувати посилання на покажчик.

Приклад заголовка функції:

```
void addUnit(  
    string*& units,  
    double**& matrix,  
    int& size  
);
```

15. У функції main() залишити організацію меню, введення даних і виклики відповідних функцій.
16. Використовувати nullptr замість значень 0 або NULL для позначення нульового покажчика.
17. Після звільнення пам'яті доцільно присвоювати покажчикам значення nullptr.
18. Протестувати багаторазове додавання нових одиниць і завершення програми після розширення масивів.

Результатом інкременту має бути програма, яка динамічно зберігає назви одиниць і матрицю коефіцієнтів, дає змогу додавати нові одиниці та коректно звільняє виділену пам'ять.

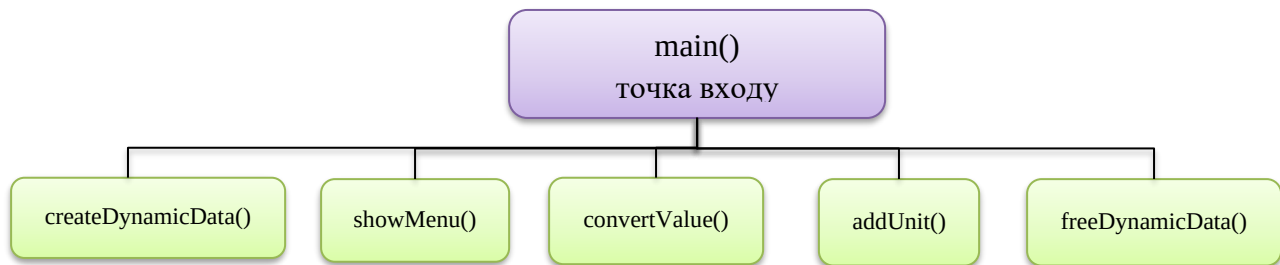


Рис. 4.15 – HIPO діаграма інкременту 8

Приклад коду та результату роботи

```
/*  
 * Проект: Програмна система конвертації одиниць вимірювання  
 * Інкремент: 8 – покажчики, посилання, масиви покажчиків,  
 * покажчики на функції та динамічна пам'ять  
 *  
 * ДОДАНО В ЦЬОМУ ІНКРЕМЕНТІ:  
 * - динамічний масив назв одиниць char**;  
 * - динамічний масив коефіцієнтів double*;  
 * - динамічну матрицю коефіцієнтів double**;  
 * - додавання нової одиниці з розширенням усіх масивів;  
 * - передавання покажчиків за посиланням;  
 * - таблицю покажчиків на функції для базових конвертацій.  
 *  
 * РЕФАКТОРИНГ ПОПЕРЕДНЬОГО ІНКРЕМЕНТУ:  
 * - фіксовані масиви й матрицю замінено динамічними;  
 * - виділення, копіювання, перебудову та звільнення пам'яті винесено  
 * в окремі функції;  
 * - main() не керує ресурсами безпосередньо.  
 *  
 * НАВЧАЛЬНЕ СПРОЩЕННЯ:
```

```

* Ручне керування пам'яттю використано навмисно для вивчення new[]/delete[].
* Назви одиниць зберігаються як C-рядки, тому код копіювання є низькорівневим.
*
* ПЛАНОВИЙ РЕФАКТОРИНГ:
* В інкременті 9 C-рядки буде замінено на std::string і додано пошук
* одиниць за текстовими назвами. В інкременті 10 паралельні масиви буде
* об'єднано в масив структур Unit.
*/

```

```

#include <locale>
#include <cstring>
#include <iomanip>
#include <iostream>
#include <limits>

namespace
{
    constexpr int INITIAL_UNIT_COUNT = 5;
    constexpr int MAX_DYNAMIC_VALUES = 100;
    constexpr double CELSIUS_TO_FAHRENHEIT_SCALE = 9.0 / 5.0;
    constexpr double FAHRENHEIT_OFFSET = 32.0;

    using ConversionFunction = double (*)(double);

    void clearInputStream()
    {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }

    bool readInt(const char prompt[], int& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }
        std::cout << "Помилка: потрібно ввести ціле число.\n";
        clearInputStream();
        return false;
    }

    bool readDouble(const char prompt[], double& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }
        std::cout << "Помилка: потрібно ввести число.\n";
        clearInputStream();
        return false;
    }
}

```

```

bool readChoice(const char prompt[], int minimum, int maximum, int& value)
{
    if (!readInt(prompt, value))
    {
        return false;
    }
    if (value < minimum || value > maximum)
    {
        std::cout << "Помилка: допустимий діапазон "
                    << minimum << ".." << maximum << ".\n";
        return false;
    }
    return true;
}

bool readRepeatAnswer()
{
    char answer = 'n';
    std::cout << "Бажаєте виконати ще одну операцію? (y/n): ";
    if (!(std::cin >> answer))
    {
        clearInputStream();
        return false;
    }
    return answer == 'y' || answer == 'Y';
}

char* copyText(const char source[])
{
    const std::size_t length = std::strlen(source);
    char* destination = new char[length + 1];
    std::strcpy(destination, source);
    return destination;
}

double** allocateMatrix(int size)
{
    double** matrix = new double* [size];
    for (int row = 0; row < size; ++row)
    {
        matrix[row] = new double[size] {};
    }
    return matrix;
}

void deleteMatrix(double**& matrix, int size)
{
    if (matrix == nullptr)
    {
        return;
    }
    for (int row = 0; row < size; ++row)

```

```

        {
            delete[] matrix[row];
        }
        delete[] matrix;
        matrix = nullptr;
    }

void deleteUnitNames(char**& unitNames, int count)
{
    if (unitNames == nullptr)
    {
        return;
    }
    for (int index = 0; index < count; ++index)
    {
        delete[] unitNames[index];
    }
    delete[] unitNames;
    unitNames = nullptr;
}

void rebuildMatrix(const double factorsToMillimeters[],
    double** matrix,
    int count)
{
    for (int source = 0; source < count; ++source)
    {
        for (int target = 0; target < count; ++target)
        {
            matrix[source][target] =
                factorsToMillimeters[source] / factorsToMillimeters[target];
        }
    }
}

void initializeData(char**& unitNames,
    double*& factorsToMillimeters,
    double**& matrix,
    int& count)
{
    const char* initialNames[INITIAL_UNIT_COUNT] = { "мм", "см", "дм", "м", "км"

};

    const double initialFactors[INITIAL_UNIT_COUNT] =
    { 1.0, 10.0, 100.0, 1000.0, 1000000.0 };

    count = INITIAL_UNIT_COUNT;
    unitNames = new char* [count];
    factorsToMillimeters = new double[count];

    for (int index = 0; index < count; ++index)
    {
        unitNames[index] = copyText(initialNames[index]);
        factorsToMillimeters[index] = initialFactors[index];
    }
}

```

```

    }

    matrix = allocateMatrix(count);
    rebuildMatrix(factorsToMillimeters, matrix, count);
}

void freeData(char**& unitNames,
              double*& factorsToMillimeters,
              double**& matrix,
              int& count)
{
    deleteUnitNames(unitNames, count);
    delete[] factorsToMillimeters;
    factorsToMillimeters = nullptr;
    deleteMatrix(matrix, count);
    count = 0;
}

void printUnits(char* const unitNames[],
               const double factorsToMillimeters[],
               int count)
{
    std::cout << "Одиниці довжини:\n";
    for (int index = 0; index < count; ++index)
    {
        std::cout << index << " - " << unitNames[index]
                  << " (1 " << unitNames[index] << " = "
                  << factorsToMillimeters[index] << " мм)\n";
    }
}

void printMatrix(char* const unitNames[], double* const matrix[], int count)
{
    std::cout << std::setw(9) << "";
    for (int column = 0; column < count; ++column)
    {
        std::cout << std::setw(12) << unitNames[column];
    }
    std::cout << '\n';

    for (int row = 0; row < count; ++row)
    {
        std::cout << std::setw(9) << unitNames[row];
        for (int column = 0; column < count; ++column)
        {
            std::cout << std::setw(12) << matrix[row][column];
        }
        std::cout << '\n';
    }
}

bool readUnitIndex(const char prompt[], int count, int& index)
{

```

```

        return readChoice(prompt, 0, count - 1, index);
    }

void convertWithMatrix(char* const unitNames[],
    double* const matrix[],
    int count)
{
    std::cout << "Доступні одиниці:\n";
    for (int index = 0; index < count; ++index)
    {
        std::cout << index << " - " << unitNames[index] << '\n';
    }

    int source = 0;
    int target = 0;
    double value = 0.0;
    if (!readDouble("Значення: ", value) ||
        !readUnitIndex("Початкова одиниця: ", count, source) ||
        !readUnitIndex("Кінцева одиниця: ", count, target))
    {
        return;
    }

    std::cout << "Результат: " << value * matrix[source][target]
        << ' ' << unitNames[target] << '\n';
}

void addUnit(char**& unitNames,
    double*& factorsToMillimeters,
    double**& matrix,
    int& count)
{
    char nameBuffer[64] = {};
    double factor = 0.0;

    std::cout << "Назва нової одиниці (без пробілів): ";
    if (!(std::cin >> std::setw(64) >> nameBuffer))
    {
        std::cout << "Помилка введення назви.\n";
        clearInputStream();
        return;
    }
    if (!readDouble("Кількість міліметрів в одній новій одиниці: ", factor) ||
        factor <= 0.0)
    {
        std::cout << "Помилка: коефіцієнт має бути додатним.\n";
        return;
    }

    const int newCount = count + 1;
    char** newNames = new char* [newCount];
    double* newFactors = new double[newCount];

```

```

    for (int index = 0; index < count; ++index)
    {
        newNames[index] = copyText(unitNames[index]);
        newFactors[index] = factorsToMillimeters[index];
    }
    newNames[count] = copyText(nameBuffer);
    newFactors[count] = factor;

    double** newMatrix = allocateMatrix(newCount);
    rebuildMatrix(newFactors, newMatrix, newCount);

    deleteUnitNames(unitNames, count);
    delete[] factorsToMillimeters;
    deleteMatrix(matrix, count);

    unitNames = newNames;
    factorsToMillimeters = newFactors;
    matrix = newMatrix;
    count = newCount;

    std::cout << "Одиницю додано.\n";
}

// === ДОДАНО В ІНКРЕМЕНТІ 8: ПОКАЖЧИКИ НА ФУНКЦІЇ ===
double metersToKilometers(double value)
{
    return value / 1000.0;
}

double kilometersToMeters(double value)
{
    return value * 1000.0;
}

double gramsToKilograms(double value)
{
    return value / 1000.0;
}

double kilogramsToGrams(double value)
{
    return value * 1000.0;
}

double celsiusToFahrenheit(double value)
{
    return value * CELSIUS_TO_FAHRENHEIT_SCALE + FAHRENHEIT_OFFSET;
}

double fahrenheitToCelsius(double value)
{
    return (value - FAHRENHEIT_OFFSET) / CELSIUS_TO_FAHRENHEIT_SCALE;
}

```

```

void runFunctionPointerDemo()
{
    const char* operationNames[] =
    {
        "Метри -> кілометри",
        "Кілометри -> метри",
        "Грами -> кілограми",
        "Кілограми -> грами",
        "Цельсій -> Фаренгейт",
        "Фаренгейт -> Цельсій"
    };
    const char* resultUnits[] = { "км", "м", "кг", "г", "°F", "°C" };
    ConversionFunction operations[] =
    {
        metersToKilometers,
        kilometersToMeters,
        gramsToKilograms,
        kilogramsToGrams,
        celsiusToFahrenheit,
        fahrenheitToCelsius
    };
    constexpr int OPERATION_COUNT = 6;

    for (int index = 0; index < OPERATION_COUNT; ++index)
    {
        std::cout << index + 1 << " - " << operationNames[index] << '\n';
    }

    int choice = 0;
    double value = 0.0;
    if (!readChoice("Операція: ", 1, OPERATION_COUNT, choice) ||
        !readDouble("Значення: ", value))
    {
        return;
    }

    const int operationIndex = choice - 1;
    std::cout << "Результат: " << operations[operationIndex](value)
        << ' ' << resultUnits[operationIndex] << '\n';
}

void processDynamicValueArray(double factor)
{
    int count = 0;
    if (!readChoice("Кількість значень (1..100): ", 1, MAX_DYNAMIC_VALUES,
count))
    {
        return;
    }

    double* sourceValues = new double[count];
    double* convertedValues = new double[count];

```

```

for (int index = 0; index < count; ++index)
{
    std::cout << "Значення [" << index << "]: ";
    while (!(std::cin >> sourceValues[index]))
    {
        std::cout << "Помилка. Повторіть: ";
        clearInputStream();
    }
    convertedValues[index] = sourceValues[index] * factor;
}

for (int index = 0; index < count; ++index)
{
    std::cout << sourceValues[index] << " -> "
        << convertedValues[index] << '\n';
}

delete[] convertedValues;
delete[] sourceValues;
}

void runApplicationMenu(char**& unitNames,
    double*& factorsToMillimeters,
    double**& matrix,
    int& count)
{
    std::cout << "=====\n";
    std::cout << "  КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ\n";
    std::cout << "=====\n";
    std::cout << "1 - Базова конвертація через покажчик на функцію\n";
    std::cout << "2 - Конвертація через динамічну матрицю\n";
    std::cout << "3 - Додати одиницю\n";
    std::cout << "4 - Показати одиниці\n";
    std::cout << "5 - Показати матрицю\n";
    std::cout << "6 - Обробити динамічний масив значень\n";

    int choice = 0;
    if (!readChoice("Ваш вибір: ", 1, 6, choice))
    {
        return;
    }

    switch (choice)
    {
    case 1:
        runFunctionPointerDemo();
        break;
    case 2:
        convertWithMatrix(unitNames, matrix, count);
        break;
    case 3:
        addUnit(unitNames, factorsToMillimeters, matrix, count);

```

```

        break;
    case 4:
        printUnits(unitNames, factorsToMillimeters, count);
        break;
    case 5:
        printMatrix(unitNames, matrix, count);
        break;
    case 6:
    {
        int source = 0;
        int target = 0;
        for (int index = 0; index < count; ++index)
        {
            std::cout << index << " - " << unitNames[index] << '\n';
        }
        if (readUnitIndex("Початкова одиниця: ", count, source) &&
            readUnitIndex("Кінцева одиниця: ", count, target))
        {
            processDynamicValueArray(matrix[source][target]);
        }
        break;
    }
}

int runApplication()
{
    char** unitNames = nullptr;
    double* factorsToMillimeters = nullptr;
    double** matrix = nullptr;
    int unitCount = 0;

    initializeData(unitNames, factorsToMillimeters, matrix, unitCount);

    do
    {
        runApplicationMenu(unitNames,
            factorsToMillimeters,
            matrix,
            unitCount);
        std::cout << '\n';
    } while (readRepeatAnswer());

    freeData(unitNames, factorsToMillimeters, matrix, unitCount);
    return 0;
}
} // namespace

int main()
{
    std::setlocale(LC_ALL, "");
    return runApplication();
}

```

```

=====
      КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ
=====
Введіть значення: 55

Оберіть категорію:
1 - Довжина
2 - Маса
3 - Температура
4 - Час
1

Конвертація довжини:
1 - Метри -> Кілометри
2 - Таблиця 1-10
3 - Рекурсивна конвертація
4 - Через масив
5 - Матриця коефіцієнтів
6 - Додати нову одиницю
6
Назва нової одиниці: decymetr
Скільки decymetr в 1 мм: 0.01
Одиницю додано!
Продовжити? (y/n): y

=====
      КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ
=====
Введіть значення: 55

Оберіть категорію:
1 - Довжина
2 - Маса
3 - Температура
4 - Час
1

Конвертація довжини:
1 - Метри -> Кілометри
2 - Таблиця 1-10
3 - Рекурсивна конвертація
4 - Через масив
5 - Матриця коефіцієнтів
6 - Додати нову одиницю
5
Одиниці:
0 - мм
1 - см
2 - м
3 - км
4 - decymetr
З: 4
В: 1
Результат: 55 см

```

Рис. 4.16 Приклад роботи інкременту 8

4.8.4 Контрольні запитання

1. Дати поняття посилальних типів даних?
2. Як здійснюється до доступ до значення змінної?
3. Як оголосити покажчик на певний тип та посилання на змінну?
4. Як ініціалізувати покажчик та посилання?
5. Що відбувається під час звернення до неініціалізованого покажчика?
6. Які значення можна присвоювати покажчику?
7. Що таке «розименування покажчика»?
8. Як змінюється значення покажчика при додаванні чи відніманні цілого числа?
9. У чому полягає особливість покажчиків типу `void*`?
10. Як використовують покажчики на функції?

4.9 Лабораторна робота №9. Обробка рядків в мові C++

4.9.1 Мета та завдання

Мета роботи:

1. Ознайомитися з поняттям рядків у мові C++ та способами їх представлення.

2. Вивчити основні методи обробки рядкових даних за допомогою стандартної бібліотеки.
3. Набути практичних навичок роботи з рядками як об'єктами класу `string`.
4. Засвоїти алгоритми пошуку, порівняння та перетворення рядків.

Завдання роботи:

1. Вивчити способи оголошення та ініціалізації рядків у C++ (`char[]` та `std::string`).
2. Ознайомитися з основними функціями та методами обробки рядків (`length`, `substr`, `find`, тощо).
3. Розробити програму для введення та виведення рядкових даних.
4. Виконати обробку рядка відповідно до заданого алгоритму (підрахунок символів, слів, заміна символів тощо).
5. Провести тестування програми на різних вхідних даних та проаналізувати результати.

4.9.2 Теоретичні відомості

Рядок – це масив символів, який закінчується символом кінця рядка `'\0'`. Значенням символа є ціле число, яке є кодом таблиці символів ASCII. Символ відображається у одинарних лапках, наприклад: `'z'`. Кожний символ в оперативній пам'яті займає один байт. Рядок характеризується довжиною, яка дорівнює кількості символів, та обсягом пам'яті. Обсяг пам'яті, який займає рядок, на одиницю більший за довжину рядка за рахунок символа кінця рядка `'\0'`.

Кожний символ рядка має свій індекс. Доступ до символа рядка здійснюється за допомогою операції `[ ]`. Рядок і всі його символи мають одне ім'я. При оголошенні рядка задаються ім'я та кількість елементів рядка. Довжина рядка визначається читанням символа кінця рядка `'\0'` під час введення рядка. Для позначення нового рядка використовується символ `'\n'`.

Рядок доступний через покажчик (адресу) на перший символ у рядку. Таким чином, ім'я рядка - це покажчик на перший символ.

Формати оголошення рядка:

```
char ім'я_рядка[кількість символів];  
char *ім'я_рядка;
```

Рядок можна ініціалізувати у процесі оголошення через покажчик на рядок.

Приклад:

```
char string[50];  
char* pointer = "mastering Visual C++"; //покажчик на рядок
```

Рядкові (літеральні) константи на відміну від значення символа записуються у подвійних лапках.

У процесі оголошення рядка як масиву символів останній повинний мати такий розмір, щоб зберігати рядок і символ кінця рядка `'\0'`. Якщо рядок більший за масив символів, то символи, які виходять за його межі, будуть змінювати дані, що знаходяться в наступних за масивом комітках пам'яті.

Рядки можуть компонуватися у масив рядків та утворювати текст. Під час оголошення масиву рядків необхідно визначити два індекси: кількість рядків і максимальну кількість символів у одному рядку.

Приклад: `char text[5][25];`

Уведення та виведення рядка

Уведення та виведення рядка здійснюється за допомогою функцій `puts(char* s);` `gets(char *s);` без використання циклів. Параметром функцій введення та виведення є покажчик на рядок. Якщо вводити рядки за допомогою класів потоків через операцію `>>`, то символи рядка вводяться до першого символа пропуску. Символи рядка після символа пропуску ігноруються.

Для рядків не визначені операції присвоєння (=) та порівняння (>, <, ==, !=). Для виконання подібних дій використовуються відповідні функції, прототипи яких визначені у файлі `string.h`. Перелік деяких функцій обробки рядків подано у таблиці.

Деякі функції обробки рядків

Умовні позначення: `destination` - рядок призначення; `source` - початковий рядок; `maxlen` - максимальна довжина рядка (підрядка); `n` - кількість символів; `s`, `s1`, `s2` - поточні рядки; `c` - символ для пошуку; `endptr` - помилка під час перетворення рядка; `radix` - основа системи числення.

Таблиця 4.7 – Прототипи функцій з файлів `string.h`, `stdlib.h`

Назва	Призначення	Формат	Значення, що повертається
<code>strupr</code>	Перетворення малих літер на великі	<code>char *strupr(char *s);</code>	Покажчик на змінений рядок
<code>strlwr</code>	Перетворення великих літер на малі	<code>char *strlwr(char *s);</code>	Покажчик на змінений рядок
<code>strcpy</code>	Копіювання рядка	<code>char *strcpy(char *destination, const char *source);</code>	Покажчик на рядок-копію
<code>strncpy</code>	Копіювання частини рядка	<code>char *strncpy(char *destination, const char *source, size_t maxlen);</code>	Покажчик на рядок-копію
<code>strlen</code>	Довжина рядка	<code>size_t strlen(const char *s);</code>	Кількість символів у рядку без урахування символа <code>'\0'</code>
<code>strcat</code>	Конкатенація рядків(об'єднання)	<code>char *strcat(char *destination, const char *source);</code>	Покажчик на об'єднаний рядок
<code>strncat</code>	Додавання частини рядка до іншого рядка	<code>char *strncat(char *destination, const char *source, size_t maxlen);</code>	Покажчик на об'єднаний рядок
<code>strtod</code>	Перетворення рядка у число типу <code>double</code>	<code>double strtod(const char *s, char **endptr);</code>	Число типу <code>double</code>
<code>strtol</code>	Перетворення рядка у число типу <code>long</code>	<code>long strtol(const char *s, char **endptr, int radix);</code>	Число типу <code>long</code>
<code>strchr</code>	Пошук останнього входження заданого символа	<code>char *strchr(char *s, int c);</code>	Покажчик на останнє місцезнаходження заданого символа

strstr	Пошук першого входження заданого підрядка	char *strstr(char *s1, const char *s2);	Покажчик на перше місцезнаходження заданого символу
strcspn	Пошук першого сегмента, що не містить символів з заданого набору символів	size_t strcspn(const char *s1, const char *s2);	Довжина початкової ланки s1, яка не містить символи з рядка s2
strcmpi	Порівняння рядків без розбіжності у регістрах літер	int strcmpi(const char *s1, const char *s2);	Значення 0, якщо рядки рівні, значення менше за 0, якщо перший рядок менший за другий, значення більше за 0, якщо перший рядок більший за другий
strcmp	Порівняння рядків	int strcmp(const char *s1, const char *s2);	Значення 0, якщо рядки рівні, значення менше за 0, якщо перший рядок менший за другий, значення більше за 0, якщо перший рядок більший за другий
strncmp	Порівняння частини рядка з частиною іншого рядка	int strncmp(const char *s1, const char *s2, size_t n);	Значення 0, якщо рядки рівні, значення менше за 0, якщо перший рядок менший за другий, значення більше за 0, якщо перший рядок більший за другий
strcmpi	Порівняння частин рядків без розбіжності у регістрах літер	int strcmpi(const char *s1, const char *s2, size_t n);	Значення 0, якщо рядки рівні, значення менше за 0, якщо перший рядок менший за другий, значення більше за 0, якщо перший рядок більший за другий
strdup	Копіювання рядка у створювану область пам'яті	char *strdup(const char *s);	Покажчик на область пам'яті, що містить копію рядка
strrev	Перевертання рядка	char *strrev(char *s);	Покажчик на рядок, що змінився
strnset	Заміна заданої кількості символів на вказаний символ	char *strnset(char *s, int ch, size_t n);	Покажчик на рядок, що змінився
strtok	Визначення лексем, відділених роздільниками	char *strtok(char *source, const char *separator);	Покажчик на знайдену лексему. У процесі повторного виклику функції для пошуку нових лексем перший параметр функції має бути NULL.

atoi	Перетворення рядка у ціле число	#include <stdlib.h> int atoi(const char *s);	Число, що утворилося, або 0, якщо перетворення неможливе
atof	Перетворення рядка у дійсне число	#include <math.h> double atof(const char *s);	Число, що утворилося, або 0, якщо перетворення неможливе
itoa	Перетворення цілого числа у рядок	#include <stdlib.h> char *itoa(int value, char *string, int radix);	Показчик на рядок

4.9.3 Приклад виконання

Розглянемо приклад виконання дев'ятого інкременту(лабораторної роботи).

9	Обробка рядків	Введення одиниць текстом ("km", "kg"). Пошук відповідних одиниць. Формування звітів	Текстовий інтерфейс конвертації
---	----------------	---	---------------------------------

Вимоги до написання коду

У дев'ятому інкременті необхідно розширити програму засобами обробки рядків. Вибір початкової та кінцевої одиниць вимірювання має здійснюватися шляхом введення їхніх назв.

Під час написання програми необхідно:

1. Використовувати тип string для зберігання назв одиниць вимірювання.
2. Зберігати назви доступних одиниць у масиві рядків.
3. Реалізувати окрему функцію пошуку введеної назви в масиві одиниць.
4. Передавати назву одиниці, масив рядків і його розмір через параметри функції.
5. Масив, який функція пошуку не змінює, передавати з модифікатором const.

Рекомендований заголовок функції:

```
int findUnit(const string& name, const string units[], int size);
```

6. У разі знаходження одиниці функція повинна повертати її індекс, а в разі відсутності — значення -1.
7. Порівнювати рядки за допомогою операції ==.
8. Перед виконанням конвертації перевіряти результати пошуку обох одиниць.
9. Не звертатися до матриці коефіцієнтів, якщо хоча б одну з одиниць не знайдено.
10. Виводити зрозуміле повідомлення про помилку, якщо введена назва відсутня у списку.
11. Використовувати змістовні назви змінних, наприклад fromUnit, toUnit, fromIndex, toIndex.
12. У функції main() залишити введення даних, виклик функції пошуку, перевірку результатів і виведення звіту.
13. Не дублювати алгоритм пошуку для початкової та кінцевої одиниць, а повторно викликати одну функцію.
14. Для введення назв без пробілів можна використовувати cin >>. Якщо назва може містити пробіли, потрібно використовувати getline().
15. Зберегти функціональність попередніх інкрементів і коректне звільнення динамічної пам'яті.

Результатом інкременту має бути програма, яка приймає назви одиниць у текстовій формі, знаходить їх у масиві, визначає відповідні індекси та виконує конвертацію.

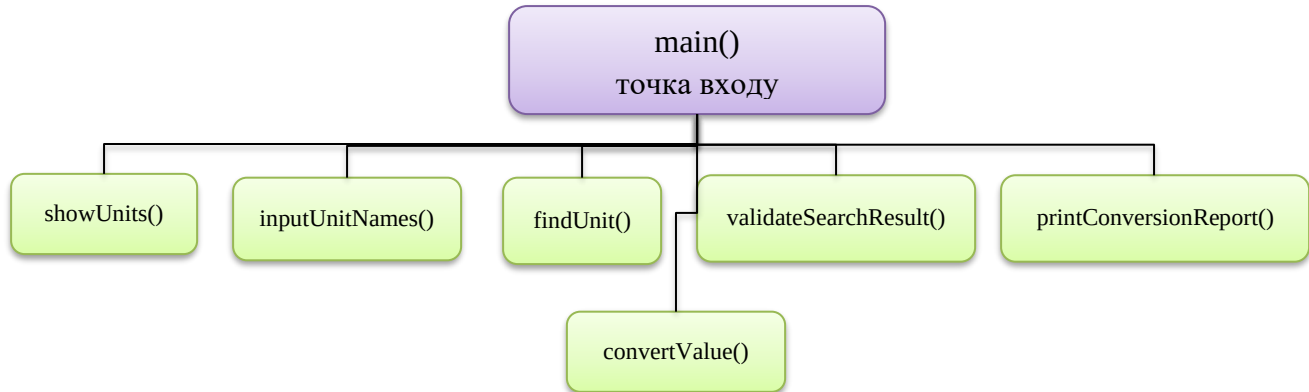


Рис. 4.17 HIPO діаграма інкременту 8

Приклад коду та результату роботи

```

/*
 * Проект:    Програмна система конвертації одиниць вимірювання
 * Інкремент: 9 – обробка рядків у C++
 *
 * ДОДАНО В ЦЬОМУ ІНКРЕМЕНТІ:
 * - зберігання назв одиниць у динамічному масиві std::string;
 * - введення назв повним рядком через getline();
 * - пошук одиниці за текстовою назвою;
 * - формування текстового звіту про конвертацію;
 * - перевірку дублювання назви під час додавання одиниці.
 *
 * РЕФАКТОРИНГ ПОПЕРЕДНЬОГО ІНКРЕМЕНТУ:
 * - char** і ручне копіювання C-рядків замінено на std::string*;
 * - усунуто функції copyText() і deleteUnitNames();
 * - сигнатура пошуку використовує const std::string& та const-масив;
 * - ручне керування пам'яттю залишено лише для навчальної динаміки масивів.
 *
 * НАВЧАЛЬНЕ СПРОЩЕННЯ:
 * Назви порівнюються без урахування регістру лише для латинських символів.
 * Дані все ще зберігаються у паралельних масивах names[] і factors[].
 *
 * ПЛАНОВИЙ РЕФАКТОРИНГ:
 * В інкременті 10 назву і коефіцієнт буде об'єднано у структуру Unit,
 * що усуне ризик розсинхронізації паралельних масивів.
 */

```

```

#include <locale>
#include <cctype>
#include <iomanip>
#include <iostream>
#include <limits>
#include <sstream>
#include <string>

```

```

namespace

```

```

{
    constexpr int INITIAL_UNIT_COUNT = 5;
    constexpr int MAX_DYNAMIC_VALUES = 100;
    constexpr double CELSIUS_TO_FAHRENHEIT_SCALE = 9.0 / 5.0;
    constexpr double FAHRENHEIT_OFFSET = 32.0;

    using ConversionFunction = double (*)(double);

    void clearInputStream()
    {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }

    bool readInt(const char prompt[], int& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }
        std::cout << "Помилка: потрібно ввести ціле число.\n";
        clearInputStream();
        return false;
    }

    bool readDouble(const char prompt[], double& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }
        std::cout << "Помилка: потрібно ввести число.\n";
        clearInputStream();
        return false;
    }

    bool readChoice(const char prompt[], int minimum, int maximum, int& value)
    {
        if (!readInt(prompt, value))
        {
            return false;
        }
        if (value < minimum || value > maximum)
        {
            std::cout << "Помилка: допустимий діапазон "
                << minimum << ".." << maximum << ".\n";
            return false;
        }
        return true;
    }
}

```

```

bool readLine(const char prompt[], std::string& value)
{
    std::cout << prompt;
    std::getline(std::cin >> std::ws, value);
    if (value.empty())
    {
        std::cout << "Помилка: рядок не може бути порожнім.\n";
        return false;
    }
    return true;
}

bool readRepeatAnswer()
{
    char answer = 'n';
    std::cout << "Бажаєте виконати ще одну операцію? (y/n): ";
    if (!(std::cin >> answer))
    {
        clearInputStream();
        return false;
    }
    return answer == 'y' || answer == 'Y';
}

std::string normalizeName(const std::string& name)
{
    std::string normalized = name;
    for (char& symbol : normalized)
    {
        const unsigned char safeSymbol = static_cast<unsigned char>(symbol);
        symbol = static_cast<char>(std::tolower(safeSymbol));
    }
    return normalized;
}

int findUnit(const std::string& name,
             const std::string unitNames[],
             int count)
{
    const std::string normalizedName = normalizeName(name);
    for (int index = 0; index < count; ++index)
    {
        if (normalizeName(unitNames[index]) == normalizedName)
        {
            return index;
        }
    }
    return -1;
}

double** allocateMatrix(int size)
{
    double** matrix = new double* [size];

```

```

        for (int row = 0; row < size; ++row)
        {
            matrix[row] = new double[size] {};
        }
        return matrix;
    }

void deleteMatrix(double**& matrix, int size)
{
    if (matrix == nullptr)
    {
        return;
    }
    for (int row = 0; row < size; ++row)
    {
        delete[] matrix[row];
    }
    delete[] matrix;
    matrix = nullptr;
}

void rebuildMatrix(const double factorsToMillimeters[],
    double** matrix,
    int count)
{
    for (int source = 0; source < count; ++source)
    {
        for (int target = 0; target < count; ++target)
        {
            matrix[source][target] =
                factorsToMillimeters[source] / factorsToMillimeters[target];
        }
    }
}

void initializeData(std::string& unitNames,
    double*& factorsToMillimeters,
    double**& matrix,
    int& count)
{
    const std::string initialNames[INITIAL_UNIT_COUNT] =
        { "mm", "cm", "dm", "m", "km" };
    const double initialFactors[INITIAL_UNIT_COUNT] =
        { 1.0, 10.0, 100.0, 1000.0, 1000000.0 };

    count = INITIAL_UNIT_COUNT;
    unitNames = new std::string[count];
    factorsToMillimeters = new double[count];
    for (int index = 0; index < count; ++index)
    {
        unitNames[index] = initialNames[index];
        factorsToMillimeters[index] = initialFactors[index];
    }
}

```

```

        matrix = allocateMatrix(count);
        rebuildMatrix(factorsToMillimeters, matrix, count);
    }

void freeData(std::string& unitNames,
             double& factorsToMillimeters,
             double**& matrix,
             int& count)
{
    delete[] unitNames;
    unitNames = nullptr;
    delete[] factorsToMillimeters;
    factorsToMillimeters = nullptr;
    deleteMatrix(matrix, count);
    count = 0;
}

void printUnits(const std::string unitNames[],
               const double factorsToMillimeters[],
               int count)
{
    std::cout << "Одиниці довжини:\n";
    for (int index = 0; index < count; ++index)
    {
        std::cout << "- " << unitNames[index]
                  << ": 1 " << unitNames[index] << " = "
                  << factorsToMillimeters[index] << " mm\n";
    }
}

void printMatrix(const std::string unitNames[],
                 double* const matrix[],
                 int count)
{
    std::cout << std::setw(10) << "";
    for (int column = 0; column < count; ++column)
    {
        std::cout << std::setw(12) << unitNames[column];
    }
    std::cout << '\n';

    for (int row = 0; row < count; ++row)
    {
        std::cout << std::setw(10) << unitNames[row];
        for (int column = 0; column < count; ++column)
        {
            std::cout << std::setw(12) << matrix[row][column];
        }
        std::cout << '\n';
    }
}

```

```

std::string buildConversionReport(double value,
    const std::string& sourceName,
    const std::string& targetName,
    double result)
{
    std::ostringstream report;
    report << "=====\n"
        << "ЗВІТ ПРО КОНВЕРТАЦІЮ\n"
        << "Початкове значення: " << value << ' ' << sourceName << '\n'
        << "Кінцева одиниця: " << targetName << '\n'
        << "Результат: " << result << ' ' << targetName << '\n'
        << "=====\n";
    return report.str();
}

void convertByNames(const std::string unitNames[],
    double* const matrix[],
    int count)
{
    double value = 0.0;
    std::string sourceName;
    std::string targetName;

    std::cout << "Доступні одиниці:\n";
    for (int index = 0; index < count; ++index)
    {
        std::cout << unitNames[index] << (index + 1 == count ? '\n' : ' ');
    }

    if (!readDouble("Значення: ", value) ||
        !readLine("З якої одиниці: ", sourceName) ||
        !readLine("У яку одиницю: ", targetName))
    {
        return;
    }

    const int sourceIndex = findUnit(sourceName, unitNames, count);
    const int targetIndex = findUnit(targetName, unitNames, count);
    if (sourceIndex < 0 || targetIndex < 0)
    {
        std::cout << "Помилка: одну або обидві одиниці не знайдено.\n";
        return;
    }

    const double result = value * matrix[sourceIndex][targetIndex];
    std::cout << buildConversionReport(value,
        unitNames[sourceIndex],
        unitNames[targetIndex],
        result);
}

void addUnit(std::string& unitNames,
    double* factorsToMillimeters,

```

```

        double**& matrix,
        int& count)
    {
        std::string newName;
        double factor = 0.0;
        if (!readLine("Назва нової одиниці: ", newName))
        {
            return;
        }
        if (findUnit(newName, unitNames, count) >= 0)
        {
            std::cout << "Помилка: одиниця з такою назвою вже існує.\n";
            return;
        }
        if (!readDouble("Кількість міліметрів в одній новій одиниці: ", factor) ||
            factor <= 0.0)
        {
            std::cout << "Помилка: коефіцієнт має бути додатним.\n";
            return;
        }

        const int newCount = count + 1;
        std::string* newNames = new std::string[newCount];
        double* newFactors = new double[newCount];
        for (int index = 0; index < count; ++index)
        {
            newNames[index] = unitNames[index];
            newFactors[index] = factorsToMillimeters[index];
        }
        newNames[count] = newName;
        newFactors[count] = factor;

        double** newMatrix = allocateMatrix(newCount);
        rebuildMatrix(newFactors, newMatrix, newCount);

        delete[] unitNames;
        delete[] factorsToMillimeters;
        deleteMatrix(matrix, count);

        unitNames = newNames;
        factorsToMillimeters = newFactors;
        matrix = newMatrix;
        count = newCount;
        std::cout << "Одиницю додано.\n";
    }

double metersToKilometers(double value)
{
    return value / 1000.0;
}

double kilometersToMeters(double value)
{

```

```

        return value * 1000.0;
    }

double celsiusToFahrenheit(double value)
{
    return value * CELSIUS_TO_FAHRENHEIT_SCALE + FAHRENHEIT_OFFSET;
}

double fahrenheitToCelsius(double value)
{
    return (value - FAHRENHEIT_OFFSET) / CELSIUS_TO_FAHRENHEIT_SCALE;
}

void runFunctionPointerDemo()
{
    const std::string operationNames[] =
    {
        "Метри -> кілометри",
        "Кілометри -> метри",
        "Цельсій -> Фаренгейт",
        "Фаренгейт -> Цельсій"
    };
    const char* resultUnits[] = { "км", "м", "°F", "°C" };
    ConversionFunction operations[] =
    {
        metersToKilometers,
        kilometersToMeters,
        celsiusToFahrenheit,
        fahrenheitToCelsius
    };
    constexpr int OPERATION_COUNT = 4;

    for (int index = 0; index < OPERATION_COUNT; ++index)
    {
        std::cout << index + 1 << " - " << operationNames[index] << '\n';
    }

    int choice = 0;
    double value = 0.0;
    if (!readChoice("Операція: ", 1, OPERATION_COUNT, choice) ||
        !readDouble("Значення: ", value))
    {
        return;
    }

    const int index = choice - 1;
    std::cout << "Результат: " << operations[index](value)
        << ' ' << resultUnits[index] << '\n';
}

void processDynamicValueArray(double factor)
{
    int count = 0;

```

```

count))
    if (!readChoice("Кількість значень (1..100): ", 1, MAX_DYNAMIC_VALUES,
    {
        return;
    }

    double* values = new double[count];
    for (int index = 0; index < count; ++index)
    {
        std::cout << "Значення [" << index << "]: ";
        while (!(std::cin >> values[index]))
        {
            std::cout << "Помилка. Повторіть: ";
            clearInputStream();
        }
    }

    for (int index = 0; index < count; ++index)
    {
        std::cout << values[index] << " -> " << values[index] * factor << '\n';
    }
    delete[] values;
}

void runMenu(std::string& unitNames,
double*& factorsToMillimeters,
double**& matrix,
int& count)
{
    std::cout << "=====\n";
    std::cout << "  КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ\n";
    std::cout << "=====\n";
    std::cout << "1 - Конвертація за текстовими назвами\n";
    std::cout << "2 - Додати одиницю\n";
    std::cout << "3 - Показати одиниці\n";
    std::cout << "4 - Показати матрицю\n";
    std::cout << "5 - Базова конвертація через покажчик на функцію\n";
    std::cout << "6 - Обробити динамічний масив значень\n";

    int choice = 0;
    if (!readChoice("Ваш вибір: ", 1, 6, choice))
    {
        return;
    }

    switch (choice)
    {
    case 1:
        convertByNames(unitNames, matrix, count);
        break;
    case 2:
        addUnit(unitNames, factorsToMillimeters, matrix, count);
        break;

```

```

        case 3:
            printUnits(unitNames, factorsToMillimeters, count);
            break;
        case 4:
            printMatrix(unitNames, matrix, count);
            break;
        case 5:
            runFunctionPointerDemo();
            break;
        case 6:
        {
            std::string sourceName;
            std::string targetName;
            if (!readLine("Початкова одиниця: ", sourceName) ||
                !readLine("Кінцева одиниця: ", targetName))
            {
                return;
            }
            const int source = findUnit(sourceName, unitNames, count);
            const int target = findUnit(targetName, unitNames, count);
            if (source < 0 || target < 0)
            {
                std::cout << "Одиницю не знайдено.\n";
                return;
            }
            processDynamicValueArray(matrix[source][target]);
            break;
        }
    }

int runApplication()
{
    std::string* unitNames = nullptr;
    double* factorsToMillimeters = nullptr;
    double** matrix = nullptr;
    int unitCount = 0;
    initializeData(unitNames, factorsToMillimeters, matrix, unitCount);

    do
    {
        runMenu(unitNames, factorsToMillimeters, matrix, unitCount);
        std::cout << '\n';
    } while (readRepeatAnswer());

    freeData(unitNames, factorsToMillimeters, matrix, unitCount);
    return 0;
}

} // namespace

int main()
{
    std::setlocale(LC_ALL, "");

```

```

    return runApplication();
}

```

```

=====
КОНВЕРТЕР ОДИНИЦЬ
=====
Введіть значення: 1500
Категорія:
1 - Довжина
Ваш вибір: 1

1 - Таблична конвертація
2 - Додати нову одиницю
3 - Ввести одиниці текстом
3
Доступні одиниці: mm cm m km
З (текст): km
В (текст): mm
=====
ЗВІТ КОНВЕРТАЦІЇ
Значення: 1500
З: km
В: mm
Результат: 1.5e+09
=====

```

Рис. 4.18 Приклад роботи інкременту 9

4.9.4 Контрольні запитання

1. Чим відрізняється поточна довжина рядка від його загальної довжини?
2. Як позначається кінець рядка?
3. Як ініціалізувати рядок під час його оголошення?
4. Як здійснити доступ до елемента рядка?
5. Як увести та вивести рядок?
6. Які бібліотечні функції визначені для змінних рядкового типу?
7. Чи можна виконати операції присвоєння рядків?
8. Як реалізуються операції порівняння рядків?
9. У чому особливість застосування функції розкладання рядка на лексеми?
10. У чому полягає різниця між `char *s1 = "string"` та `char s2[] = "string"` в контексті типу, розміщення, можливості зміни?

4.10 Лабораторна робота №10. Структури, масиви структур та об'єднання

4.10.1 Мета та завдання

Мета роботи:

1. Ознайомитися з поняттям структур, масивів структур та об'єднань у мові C++.
2. Вивчити способи створення та використання користувацьких типів даних.
3. Набути практичних навичок організації складних структур даних.
4. Засвоїти особливості роботи з масивами структур та об'єднаннями.

Завдання роботи:

1. Вивчити синтаксис оголошення структур (struct) та об'єднань (union).
2. Ознайомитися зі способами ініціалізації та доступу до полів структур.
3. Розробити програму для роботи зі структурою, що містить кілька різнотипних полів.
4. Реалізувати обробку масиву структур (введення, виведення, пошук, сортування).
5. Ознайомитися з особливостями використання об'єднань та їх обмеженнями.
6. Провести тестування програми та проаналізувати результати її роботи.

4.10.2 Теоретичні відомості

Структура - це тип даних користувача, який об'єднує дані різних типів. Елементами структури можуть бути дані будь-якого типу. Елементи структури називаються полями.

Оголошення структури задається у такому вигляді:

```
struct ім'я_типу_структури
{
    тип ім'я_змінної;
    тип ім'я_змінної;
    ..... ;
};
```

Оголошення структури обов'язково закінчується фігурною дужкою }, після якої ставиться крапка з комою.

Оголошення структурного типу не резервує жодного простору в пам'яті, тільки створює новий тип даних, який використовується у процесі оголошенні змінних. Пам'ять для структури резервуватиметься під час оголошення змінної. Для визначення обсягу пам'яті, який резервується для змінної структурного типу, використовується операція sizeof(ім'я_типу_структури).

Структурна змінна оголошується як будь-яка змінна через ім'я або покажчик на тип:

```
ім'я_типу_структури ім'я_змінної;
ім'я_типу_структури *ім'я_покажчика;
```

Доступ до елемента структури здійснюється через складене ім'я, яке утворюється за допомогою операції точки через ім'я об'єкта чи посилання:

```
ім'я_змінної_типу_структури.ім'я_поля_структури
```

Якщо задано покажчик на структурний тип, то для доступу до елемента структури використовується складене ім'я, яке утворюється за допомогою операції стрілка:

```
ім'я_покажчика_на_тип_структури -> ім'я_поля_структури
```

У будь-якій функції можна оголошувати структурну змінну, але оголошення типу структури треба подати як глобальне на початку програми.

Приклад:

```
#include <iostream>
struct anketa //структурний тип
{
    char fio[20]; //поля структури
```

```

    int year_birth;
    float salary;
};
void main()
{ //доступ до полів структури
  anketa student, *pointer_student;
  pointer_student = &student;
  cin >> student.fio;
  cin >> pointer_student -> salary ;
  cout << student.fio;
  cout << pointer_student ->salary ;
}

```

Структурній змінній можна присвоїти значення іншої структурної змінної, можна передавати як параметри у функцію і повертати в основну програму як результат роботи функції над полями структури. Попередньо оголошений покажчик на структурну змінну передається у функцію під час її виклику, щоб не створювати копії основного об'єкта.

Якщо у функцію передається покажчик на структурну змінну, то у процесі виклику функції параметр замінюється на аргумент, значенням якого є адреса структурної змінної: *&ім'я_структурної_змінної*

Якщо параметром є структура, то під час виклику функції аргументом є ім'я структурної змінної.

Структурні змінні можна об'єднувати у масиви, кожний елемент якого - це структурна змінна. Оголошення масиву структур:

ім'я_типу_структури ім'я_масиву_структур[кількість елементів];

Доступ до поля масиву структур здійснюється за допомогою індекса елемента:

ім'я_масиву_структур[індекс].ім'я_поля

Якщо у функцію передається покажчик на масив структур, то він ініціалізується значенням нульового елемента масиву:

*ім'я_типу_структури *ім'я_показчика; //оголошення показчика*
ім'я_показчика = &ім'я_масиву_структур[0];

4.10.3 Приклад виконання

Розглянемо приклад виконання десятого інкременту(лабораторної роботи).

10	Структури, масиви структур та об'єднання	Представлення даних у вигляді структур та масивів структур	Використання масивів і структур
----	--	--	---------------------------------

Вимоги до написання коду

У десятому інкременті необхідно замінити окремі масиви назв і коефіцієнтів масивом структур. Кожна структура повинна об'єднувати пов'язані дані про одну одиницю вимірювання.

Під час написання програми необхідно:

1. Оголосити структуру Unit, що містить назву одиниці та коефіцієнт перетворення до базової одиниці.

```
struct Unit
{
    string name;
    double toMM;
};
```
2. Використовувати масив структур Unit для зберігання всіх одиниць вимірювання.
3. Забезпечити, щоб кожен елемент масиву містив повний набір пов'язаних даних про одну одиницю.
4. Для доступу до полів структурної змінної використовувати оператор ..
5. Для доступу до полів структури через покажчик використовувати оператор ->.
6. Реалізувати окрему функцію пошуку одиниці за її назвою.
Рекомендований заголовок:

```
int findUnit(
    const string& name,
    const Unit units[],
    int size
);
```
7. У разі успішного пошуку функція повинна повертати індекс елемента, а в разі відсутності одиниці — значення -1.
8. Передавати масив структур у функції разом із кількістю його елементів.
9. Масив структур, який функція не змінює, передавати з модифікатором const.
10. Реалізувати окрему функцію конвертації значення між двома одиницями.
Рекомендований заголовок:

```
double convertValue(
    double value,
    const Unit& from,
    const Unit& to
);
```
11. Виконувати конвертацію через базову одиницю:

```
double valueInMM = value * from.toMM;
double result = valueInMM / to.toMM;
```
12. Перевіряти результати пошуку одиниць до звернення до елементів масиву.
13. Не дублювати назви та коефіцієнти в окремих масивах після введення структури Unit.
14. У функції main() залишити організацію меню, введення даних, виклики функцій і виведення результатів.
15. Використовувати змістовні назви полів, змінних і функцій.
16. Не використовувати глобальний масив структур без обґрунтованої потреби.
17. Якщо в інкременті застосовується union, його поля повинні поділяти одну область пам'яті, а програма має чітко визначати, яке поле є активним. Не слід використовувати об'єднання лише формально.

18. Перевірити роботу програми для наявних і відсутніх назв одиниць, нульових, додатних і від'ємних значень.

Результатом інкременту має бути програма, у якій дані про одиниці вимірювання організовано у вигляді масиву структур, а пошук і конвертація виконуються окремими функціями.

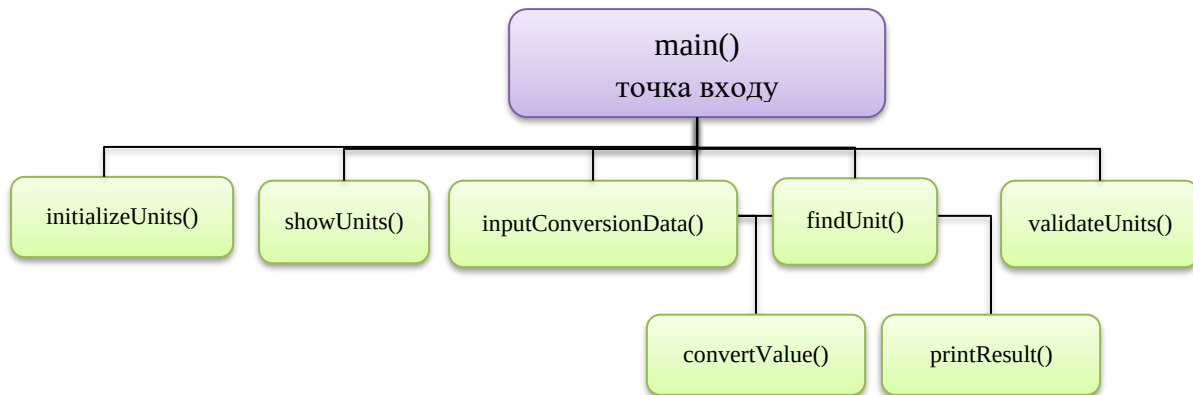


Рис. 4.19 HIPO діаграма інкременту 10

Приклад коду та результату роботи

```

/*
 * Проект:    Програмна система конвертації одиниць вимірювання
 * Інкремент: 10 – структури та масиви структур
 *
 * ДОДАНО В ЦЬОМУ ІНКРЕМЕНТІ:
 * - структуру Unit, що об'єднує назву та коефіцієнт одиниці;
 * - динамічний масив структур Unit;
 * - пошук, додавання, виведення і конвертацію через структури;
 * - передавання структур у функції через const-посилання.
 *
 * РЕФАКТОРИНГ ПОПЕРЕДНЬОГО ІНКРЕМЕНТУ:
 * - паралельні масиви unitNames[] і factors[] замінено одним масивом Unit[];
 * - матриця не зберігається як залежний стан, а обчислюється під час
 *   виведення з полів структур (DRY, низька зв'язаність);
 * - функція convertValue() відповідає лише за обчислення (SRP).
 *
 * НАВЧАЛЬНЕ СПРОЩЕННЯ:
 * Масив структур керується вручну через new[]/delete[], щоб зберегти зв'язок
 * з інкрементом 8. Об'єднання union свідомо не використовується: предметна
 * модель не містить взаємовиключних полів, тому формальне застосування union
 * суперечило б YAGNI та ускладнювало код без користі.
 *
 * ПЛАНОВИЙ РЕФАКТОРИНГ:
 * В інкременті 11 буде додано безпечне збереження і завантаження масиву Unit
 * у текстовому та бінарному форматах.
 */

#include <locale>
#include <cctype>
#include <iomanip>
#include <iostream>

```

```

#include <limits>
#include <sstream>
#include <string>

namespace
{
    constexpr int INITIAL_UNIT_COUNT = 5;
    constexpr int MAX_DYNAMIC_VALUES = 100;
    constexpr double CELSIUS_TO_FAHRENHEIT_SCALE = 9.0 / 5.0;
    constexpr double FAHRENHEIT_OFFSET = 32.0;

    struct Unit
    {
        std::string name;
        double toMillimeters = 0.0;
    };

    using ConversionFunction = double (*)(double);

    void clearInputStream()
    {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }

    bool readInt(const char prompt[], int& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }
        std::cout << "Помилка: потрібно ввести ціле число.\n";
        clearInputStream();
        return false;
    }

    bool readDouble(const char prompt[], double& value)
    {
        std::cout << prompt;
        if (std::cin >> value)
        {
            return true;
        }
        std::cout << "Помилка: потрібно ввести число.\n";
        clearInputStream();
        return false;
    }

    bool readChoice(const char prompt[], int minimum, int maximum, int& value)
    {
        if (!readInt(prompt, value))
        {

```

```

        return false;
    }
    if (value < minimum || value > maximum)
    {
        std::cout << "Помилка: допустимий діапазон "
                    << minimum << ".." << maximum << ".\n";
        return false;
    }
    return true;
}

bool readLine(const char prompt[], std::string& value)
{
    std::cout << prompt;
    std::getline(std::cin >> std::ws, value);
    if (value.empty())
    {
        std::cout << "Помилка: рядок не може бути порожнім.\n";
        return false;
    }
    return true;
}

bool readRepeatAnswer()
{
    char answer = 'n';
    std::cout << "Бажаєте виконати ще одну операцію? (y/n): ";
    if (!(std::cin >> answer))
    {
        clearInputStream();
        return false;
    }
    return answer == 'y' || answer == 'Y';
}

std::string normalizeName(const std::string& name)
{
    std::string normalized = name;
    for (char& symbol : normalized)
    {
        const unsigned char safeSymbol = static_cast<unsigned char>(symbol);
        symbol = static_cast<char>(std::tolower(safeSymbol));
    }
    return normalized;
}

void initializeUnits(Unit*& units, int& count)
{
    const Unit initialUnits[INITIAL_UNIT_COUNT] =
    {
        {"mm", 1.0},
        {"cm", 10.0},
        {"dm", 100.0},

```

```

        {"m", 1000.0},
        {"km", 1000000.0}
    };

    count = INITIAL_UNIT_COUNT;
    units = new Unit[count];
    for (int index = 0; index < count; ++index)
    {
        units[index] = initialUnits[index];
    }
}

void freeUnits(Unit*& units, int& count)
{
    delete[] units;
    units = nullptr;
    count = 0;
}

int findUnit(const std::string& name, const Unit units[], int count)
{
    const std::string normalizedName = normalizeName(name);
    for (int index = 0; index < count; ++index)
    {
        if (normalizeName(units[index].name) == normalizedName)
        {
            return index;
        }
    }
    return -1;
}

double convertValue(double value, const Unit& source, const Unit& target)
{
    const double valueInMillimeters = value * source.toMillimeters;
    return valueInMillimeters / target.toMillimeters;
}

void printUnits(const Unit units[], int count)
{
    std::cout << "Одиниці довжини:\n";
    for (int index = 0; index < count; ++index)
    {
        std::cout << "- " << units[index].name
            << ": 1 " << units[index].name << " = "
            << units[index].toMillimeters << " mm\n";
    }
}

void printConversionMatrix(const Unit units[], int count)
{
    std::cout << std::setw(10) << "";
    for (int column = 0; column < count; ++column)

```

```

    {
        std::cout << std::setw(12) << units[column].name;
    }
    std::cout << '\n';

    for (int row = 0; row < count; ++row)
    {
        std::cout << std::setw(10) << units[row].name;
        for (int column = 0; column < count; ++column)
        {
            std::cout << std::setw(12)
                << convertValue(1.0, units[row], units[column]);
        }
        std::cout << '\n';
    }
}

std::string buildConversionReport(double value,
    const Unit& source,
    const Unit& target,
    double result)
{
    std::ostringstream report;
    report << "=====\n"
        << "ЗВІТ ПРО КОНВЕРТАЦІЮ\n"
        << "Початкове значення: " << value << ' ' << source.name << '\n'
        << "Кінцева одиниця: " << target.name << '\n'
        << "Результат: " << result << ' ' << target.name << '\n'
        << "=====\n";
    return report.str();
}

void convertByNames(const Unit units[], int count)
{
    printUnits(units, count);

    double value = 0.0;
    std::string sourceName;
    std::string targetName;
    if (!readDouble("Значення: ", value) ||
        !readLine("З якої одиниці: ", sourceName) ||
        !readLine("У яку одиницю: ", targetName))
    {
        return;
    }

    const int sourceIndex = findUnit(sourceName, units, count);
    const int targetIndex = findUnit(targetName, units, count);
    if (sourceIndex < 0 || targetIndex < 0)
    {
        std::cout << "Помилка: одну або обидві одиниці не знайдено.\n";
        return;
    }
}

```

```

        const double result = convertValue(value,
            units[sourceIndex],
            units[targetIndex]);
        std::cout << buildConversionReport(value,
            units[sourceIndex],
            units[targetIndex],
            result);
    }

void addUnit(Unit*& units, int& count)
{
    Unit newUnit;
    if (!readLine("Назва нової одиниці: ", newUnit.name))
    {
        return;
    }
    if (findUnit(newUnit.name, units, count) >= 0)
    {
        std::cout << "Помилка: одиниця з такою назвою вже існує.\n";
        return;
    }
    if (!readDouble("Кількість міліметрів в одній новій одиниці: ",
        newUnit.toMillimeters) ||
        newUnit.toMillimeters <= 0.0)
    {
        std::cout << "Помилка: коефіцієнт має бути додатним.\n";
        return;
    }

    Unit* expandedUnits = new Unit[count + 1];
    for (int index = 0; index < count; ++index)
    {
        expandedUnits[index] = units[index];
    }
    expandedUnits[count] = newUnit;

    delete[] units;
    units = expandedUnits;
    ++count;
    std::cout << "Одиницю додано.\n";
}

void processDynamicValueArray(const Unit& source, const Unit& target)
{
    int count = 0;
    if (!readChoice("Кількість значень (1..100): ", 1, MAX_DYNAMIC_VALUES,
count))
    {
        return;
    }

    double* values = new double[count];

```

```

    for (int index = 0; index < count; ++index)
    {
        std::cout << "Значення [" << index << "]: ";
        while (!(std::cin >> values[index]))
        {
            std::cout << "Помилка. Повторіть: ";
            clearInputStream();
        }
    }

    for (int index = 0; index < count; ++index)
    {
        std::cout << values[index] << ' ' << source.name
            << " = " << convertValue(values[index], source, target)
            << ' ' << target.name << '\n';
    }
    delete[] values;
}

void runArrayConversion(const Unit units[], int count)
{
    std::string sourceName;
    std::string targetName;
    if (!readLine("Початкова одиниця: ", sourceName) ||
        !readLine("Кінцева одиниця: ", targetName))
    {
        return;
    }

    const int sourceIndex = findUnit(sourceName, units, count);
    const int targetIndex = findUnit(targetName, units, count);
    if (sourceIndex < 0 || targetIndex < 0)
    {
        std::cout << "Одиницю не знайдено.\n";
        return;
    }

    processDynamicValueArray(units[sourceIndex], units[targetIndex]);
}

double metersToKilometers(double value)
{
    return value / 1000.0;
}

double kilometersToMeters(double value)
{
    return value * 1000.0;
}

double celsiusToFahrenheit(double value)
{
    return value * CELSIUS_TO_FAHRENHEIT_SCALE + FAHRENHEIT_OFFSET;
}

```

```

}

double fahrenheitToCelsius(double value)
{
    return (value - FAHRENHEIT_OFFSET) / CELSIUS_TO_FAHRENHEIT_SCALE;
}

void runFunctionPointerDemo()
{
    const char* operationNames[] =
    {
        "Метри -> кілометри",
        "Кілометри -> метри",
        "Цельсій -> Фаренгейт",
        "Фаренгейт -> Цельсій"
    };
    const char* resultUnits[] = { "км", "м", "°F", "°C" };
    ConversionFunction operations[] =
    {
        metersToKilometers,
        kilometersToMeters,
        celsiusToFahrenheit,
        fahrenheitToCelsius
    };
    constexpr int OPERATION_COUNT = 4;

    for (int index = 0; index < OPERATION_COUNT; ++index)
    {
        std::cout << index + 1 << " - " << operationNames[index] << '\n';
    }

    int choice = 0;
    double value = 0.0;
    if (!readChoice("Операція: ", 1, OPERATION_COUNT, choice) ||
        !readDouble("Значення: ", value))
    {
        return;
    }

    const int index = choice - 1;
    std::cout << "Результат: " << operations[index](value)
        << ' ' << resultUnits[index] << '\n';
}

void runMenu(Unit*& units, int& count)
{
    std::cout << "=====\n";
    std::cout << "    КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ\n";
    std::cout << "=====\n";
    std::cout << "1 - Конвертація за назвами\n";
    std::cout << "2 - Додати одиницю\n";
    std::cout << "3 - Показати одиниці\n";
    std::cout << "4 - Показати матрицю коефіцієнтів\n";
}

```

```

std::cout << "5 - Обробити масив значень\n";
std::cout << "6 - Базова конвертація через покажчик на функцію\n";

int choice = 0;
if (!readChoice("Ваш вибір: ", 1, 6, choice))
{
    return;
}

switch (choice)
{
case 1:
    convertByNames(units, count);
    break;
case 2:
    addUnit(units, count);
    break;
case 3:
    printUnits(units, count);
    break;
case 4:
    printConversionMatrix(units, count);
    break;
case 5:
    runArrayConversion(units, count);
    break;
case 6:
    runFunctionPointerDemo();
    break;
}
}

int runApplication()
{
    Unit* units = nullptr;
    int unitCount = 0;
    initializeUnits(units, unitCount);

    do
    {
        runMenu(units, unitCount);
        std::cout << '\n';
    } while (readRepeatAnswer());

    freeUnits(units, unitCount);
    return 0;
}
} // namespace

int main()
{
    std::setlocale(LC_ALL, "");
    return runApplication();
}

```

}

```
=====
КОНВЕРТЕР ОДИНИЦЬ
=====
Введіть значення: 300

1 - Конвертація (структури)
2 - Додати одиницю
3 - Показати одиниці
1
Доступні одиниці: mm cm m km
З: m
В: km
Результат: 0.3 km
```

Рис. 4.20 Приклад роботи інкременту 10

4.10.4 Контрольні запитання

1. Дати поняття структурного типу даних та оголошення структурної змінної, масиву структур.
2. Як здійснюється доступ до елементів структури?
3. Як передати структуру або масив структур до функції як параметр?
4. Як повернути з функції масив структур?
5. Які операції можна виконувати із структурами?
6. Як визначається обсяг пам'яті, що потребує структура, і коли вона резервується?
7. Який розмір має порожня структура `struct Empty {}`;
8. Що станеться, якщо присвоїти одну структуру іншій (`s1 = s2`), якщо елементом структури є динамічний масив (показчик)?
9. Що станеться з даними структури, якщо вона є частиною `union`, і ми запишемо дані в інше поле `union`?
10. Чи можна оголосити структуру, розмір масиву в якій визначається під час виконання? Що визначить операція `sizeof`?

4.11 Лабораторна робота №11. Текстові та бінарні файли

4.11.1 Мета та завдання

Мета роботи:

1. Ознайомитися з принципами організації роботи з текстовими та бінарними файлами у мові C/C++.
2. Вивчити способи створення, відкриття, читання та запису файлів.
3. Набути практичних навичок використання файлових потоків для збереження та обробки даних.
4. Засвоїти відмінності між текстовим і бінарним поданням даних.

Завдання роботи:

1. Вивчити основи роботи з файловими потоками (`ifstream`, `ofstream`, `fstream`).

2. Ознайомитися з режимами відкриття файлів та їх призначенням.
3. Розробити програму для запису та читання даних у текстовий файл.
4. Реалізувати програму для роботи з бінарними файлами (запис і читання структур або масивів).
5. Забезпечити обробку можливих помилок при роботі з файлами.
6. Провести тестування програми та перевірити коректність збереження і відновлення даних.

4.11.2 Теоретичні відомості

Текстовий файл є сукупністю символьних рядків змінної довжини. Кожен рядок завершується *маркером кінця рядка* - спеціальною парою керуючих символів, коди яких: 13 (повернення каретки) та 10 (переведення рядка). Наприкінці файла записується *маркер кінця файла* - керуючий символ з кодом 26.

Обробка файлів передбачає такі операції: відкриття файлу на запис або на читання, запис даних у файл з оперативної пам'яті або читання даних з файлу в оперативну пам'ять, закриття файлу. Спочатку розглянемо функції для обробки файлів мовою C, потім використання класів потоків C++ для обробки файлів.

Відкриття і закриття файла

Відкриття файла в мові C здійснюється функцією `fopen()`. Її прототип оголошено так:

```
FILE *fopen(const char *filename, const char *mode);
```

Функція має такі параметри: *filename* - рядковий вираз, значенням якого має бути ім'я фізичного файла; *mode* - рядковий вираз, значенням якого є режим доступу до файла. Другий параметр функції `fopen()` задає режим доступу до файла та може приймати такі значення: "r" - відкрити файл для читання; "w" - створити новий файл для запису; "a" - відкрити файл для додавання даних у кінець файла або створити новий файл, якщо він не існує; "r+" - відкрити існуючий файл для читання та запису; "w+" - створити новий файл для запису та читання; "a+" - відкрити файл в режимі читання та запису для додавання нових даних в кінець файла або створити новий файл, якщо він не існує.

Файли закриваються функцією `fclose()`:

```
fclose(<показчик на файл>);
```

Функція `fclose()` повертає значення нуль у разі успішного закриття файла або EOF, якщо виникла помилка

Зчитування і запис текстових файлів

Зчитування окремих символів із текстового файла здійснюється функцією `fgetc()`, яка має прототип:

```
int fgetc(FILE *stream);
```

Зчитування текстового файла по рядках має такий прототип:

```
char *fgets(char *str, int n, FILE *stream);
```

Тут **str** - буфер, в якому запам'ятовується зчитаний з файла рядок, заданий як покажчик на тип **char**; **n** - максимальна кількість зчитаних символів; **stream** - покажчик на файл, з якого зчитується рядок.

У мові С означені функції **fputc()**, **fputs()**, що здійснюють запис символів і рядків до файлу. Вони мають такі прототипи:

```
int fputc(int c, FILE *stream);  
int fputs(const char *str, FILE *stream);
```

Тут **c** - символ, який записують до файла; **stream** - покажчик на файл; **str** - покажчик на рядок, який записують до текстового файла.

Функції зчитування текстових файлів за форматом, який визначає тип значення, що його функція має повернути, мають такі прототипи:

```
int fscanf(FILE *stream, const char *format [,argument ]...);  
int fprintf(FILE *stream, const char *format [,argument ]...);
```

Функції мають параметри: **stream** - покажчик на текстовий файл; **format** - покажчик на рядок формату, який задає правила перетворення символів; **argument** - адреси аргументів, що посилаються на змінні програми.

Для зчитування з файла всіх даних потрібно використовувати функцію **feof()**, яка визначає, чи досягнуто кінця файла:

```
int feof(FILE *stream);
```

Читання та запис бінарних файлів

Під час відкриття бінарних файлів викликають функцію **fopen()**, проте вказують режим бінарної обробки: **"rb"** - тільки зчитування, **"wb"** - тільки запис, **"ab"** - доповнення в кінець, **"r+b"** і **"w+b"** позначають режим запису та зчитування.

Приклад відкриття бінарного файлу **"MyFile.dat"** з покажчиком **FILE* fp**
fp=fopen("MyFile.dat", "r+b");

Прототипи функцій зчитування та запису в бінарні файли є такими:

```
size_t fread(void *buffer, size_t size, size_t count, FILE *stream);  
size_t fwrite(const void *buffer, size_t size, size_t count, FILE *stream);
```

Тут **buffer** - адреса змінної, з якої байти копіюються на диск під час виклику **fwrite()**, або адреса змінної, в яку байти зчитуються під час виклику **fread()**; **size** - кількість байтів, що виділені під змінну; **count** - кількість елементів розміру **size**, що записуються на диск або зчитуються з диска; **stream** - покажчик на файловий потік, відкритий у бінарному режимі.

Прямий доступ до компонентів бінарних файлів

Наведені далі функції мови С реалізують прямий метод доступу до файлу.

Функція **fgetpos()** зберігає позицію файлового покажчика у змінній, що адресується другим параметром. Значення позиції файлового покажчика дорівнює зміщенню в байтах від початку файла до позиції, з якої має розпочатися наступна операція введення-виведення. Прототип функції такий:

```
int fgetpos(FILE *stream, fpos_t *pos);
```

Аргументи: **stream** - покажчик на файловий потік; **pos** - покажчик на поточну позицію у файлі; тип **fpos_t** є перейменуванням (аліасом) типу **long**. У разі успішного виклику функції вона повертає значення нуль, інакше -1.

Функція **fsetpos()** відновлює позицію файлового покажчика, встановлюючи його рівним значенню, що його отримала функція **fgetpos()**. Параметри функції **fsetpos()** аналогічні розглянутим вище, значення **pos** визначається попереднім викликом **fgetpos()**. Функція повертає значення нуль, якщо її робота успішна та ненульове значення в разі помилки. Прототип функції **fsetpos()** оголошений так:

```
int fsetpos(FILE *stream, const fpos_t *pos);
```

Отримати поточну позицію файлового покажчика, яка згодом використовуватиметься для переміщення файлового покажчика на потрібний байт, можна за допомогою функції **ftell()** з прототипом:

```
long ftell( FILE *stream);
```

Функція **ftell()** повертає позицію файлового покажчика. Значення , що його повертає функція **ftell()**, слід передати у функцію **fseek()**, яка переміщує файловий покажчик на певну кількість байтів вперед по файлу (від його початку) або назад (від його кінця).

Функція **fseek()** має такий прототип:

```
int fseek( FILE *stream, long offset, int origin);
```

Тут **stream** - покажчик на тип **FILE**; **offset** - кількість байтів, на які слід перемістити файловий покажчик даних у напрямку, що вказаний третім параметром; **origin** - напрям переміщення файлового покажчика, задається константами:

SEEK_SET для переміщення файлового покажчика в напрямку від початку файла,

SEEK_END для переміщення файлового покажчика в напрямку від кінця файла,

SEEK_CUR для переміщення файлового покажчика відносно його поточної позиції

Використання класів потоків для роботи з файлами в мові C++

У мові C++ робота з файлами реалізується через механізм потоків (streams), який є частиною стандартної бібліотеки. Потік – це абстракція послідовності байтів, що передаються між програмою та зовнішнім джерелом (файлом, консоллю тощо). Класи потоків бібліотеки **<fstream>** забезпечують зручний, безпечний та універсальний механізм роботи з файлами у C++. Вони дозволяють виконувати як текстовий, так і бінарний ввід/вивід, контролювати стан потоку та керувати позицією у файлі, що робить їх ключовим інструментом при розробці програм, які працюють із зовнішніми даними.

Для роботи з файлами використовується заголовковий файл:

```
#include <fstream>
```

Бібліотека **<fstream>** визначає ієрархію класів, що забезпечують файловий ввід/вивід. У стандартній бібліотеці C++ визначено три основні класи для роботи з файлами: **ifstream** – клас вхідних файлових потоків, використовується для читання даних із файлу; **ofstream** – клас

вихідних файлових потоків, використовується для запису даних у файл. `fstream` — клас двонаправлених файлових потоків, забезпечує як читання, так і запис даних.

Використання файлів в C++ програмі припускає такі операції: створення потоку, відкриття потоку і скріплення його з файлом, обмін (введення/виведення) даними, знищення потоку, закриття файлу.

Відкриття файлу

Файл може бути відкритий двома способами: через конструктор

```
std::ifstream fin("input.txt");
```

За допомогою методу `open()`

```
std::ofstream fout;  
fout.open("output.txt");
```

Під час відкриття файлу потрібно вказати режим роботи (табл. 4.11):

Таблиця 4.8 – Режими роботи з файлами в C++

Режим	Опис
<code>std::ios::in</code>	відкриття для читання
<code>std::ios::out</code>	відкриття для запису
<code>std::ios::app</code>	додавання в кінець файлу
<code>std::ios::ate</code>	позиціонування в кінець після відкриття
<code>std::ios::trunc</code>	очищення файлу
<code>std::ios::binary</code>	бінарний режим

Приклад:

```
std::ofstream fout("file.txt", std::ios::out | std::ios::app);
```

Перевірка успішності відкриття

Перед роботою з файлом необхідно перевірити, чи був він успішно відкритий:

```
if (!fin.is_open())  
{  
    std::cout << "Помилка відкриття файлу\n";  
}
```

Запис у файл

Для запису у файл використовуються ті самі оператори, що і для виводу в консоль:

```
fout << "Hello, world!" << std::endl;
```

Можна записувати у файл дані різних типів:

```
int number = 10;  
char ctr[20];  
cin>>str;  
fout << number << " " << str << endl;
```

Читання з файлу

Форматоване читання

```
int number;
fin >> number;
```

Читання рядків

```
std::string line;
std::getline(fin, line);
```

Перевірка стану потоку

Потоки мають спеціальні методи для перевірки стану (табл 4.12):

Таблиця 4.9 – Методи класів потоків для перевірки стану потоків

Метод	Опис
<code>eof()</code>	Досягнуто кінець файлу
<code>fail()</code>	Помилка операції
<code>good()</code>	Потік у нормальному стані
<code>bad()</code>	Серйозна помилка

Приклад:

```
while (!fin.eof())
{
    std::string line;
    std::getline(fin, line);
}
```

Автори рекомендують у практиці краще використовувати перевірку результату `getline`.

Закриття файлу

Після завершення роботи файл слід закрити:

```
fin.close();
fout.close();
```

Закриття файлу — це обов’язкова операція, яка забезпечує повний запис даних; звільнення ресурсів; цілісність і коректність файлу. Без закриття файлу частина даних може просто не потрапити у файл. Якщо не викликати `close()`, файл буде закритий автоматично при знищенні об’єкта.

Бінарні файли в C++

Для роботи з бінарними файлами використовуються методи: `read()` та `write()`.

Приклад запису даних у бінарний файл:

```
int a = 100;
fout.write(reinterpret_cast<char*>(&a), sizeof(a));
```

Приклад читання даних з бінарного файлу:

```
fin.read(reinterpret_cast<char*>(&a), sizeof(a));
```

Тут ідентифікатор `fout` визначає об’єкт потоку виводу у файл, зазвичай типу `std::ofstream`; ідентифікатор `fin` визначає об’єкт потоку вводу з файлу, зазвичай типу `std::ifstream`.

Позиціювання у файлі

Файлові потоки дозволяють змінювати позицію читання/запису у файлі (табл. 4.10).

Таблиця 4.10 – Методи позиціювання покажчиків читання та запису у файлових потоках

Метод	Опис
<code>seekg()</code>	встановлення позиції для читання
<code>seekp()</code>	встановлення позиції для запису
<code>tellg()</code>	отримання позиції читання
<code>tellp()</code>	отримання позиції запису

Приклад:

```
fin.seekg(0, std::ios::beg);
```

4.11.3 Приклад виконання

Розглянемо приклад виконання одинадцятого інкременту(лабораторної роботи).

11	Текстові та бінарні файли	Збереження списку одиниць і коефіцієнтів. Завантаження конфігурації	Імпорт/експорт даних конвертера
----	---------------------------	---	---------------------------------

Вимоги до написання коду

В одинадцятому інкременті необхідно реалізувати збереження списку одиниць вимірювання та їхніх коефіцієнтів у файл, а також відновлення даних із файлу.

Під час написання програми необхідно:

1. Підключити заголовний файл `<fstream>`.
2. Для збереження даних використовувати об'єкт типу `ofstream`, а для завантаження — об'єкт типу `ifstream`.
3. Реалізувати окремі функції збереження та завантаження даних.
4. Передавати до функції збереження масив структур `Unit` та кількість його елементів.
5. Масив, який не змінюється під час запису, передавати з модифікатором `const`.

Рекомендований заголовок функції:

```
bool saveToFile(  
    const Unit units[],  
    int size,  
    const string& fileName  
);
```

6. Спочатку записувати у файл кількість одиниць, а потім назву й коефіцієнт кожної одиниці.
7. Після відкриття файлу перевіряти стан файлового потоку.
8. У разі помилки відкриття, читання або запису повертати з функції ознаку невдалого виконання.
9. Під час завантаження спочатку зчитувати кількість записів і перевіряти її коректність.
10. Виділяти пам'ять для нового масиву лише після успішного зчитування його розміру.
11. Не видаляти наявний масив до завершення успішного читання всіх нових даних.

12. Після успішного завантаження звільнити пам'ять старого масиву, оновити покажчик units і значення size.

Рекомендований заголовок функції:

```
bool loadFromFile(  
    Unit*& units,  
    int& size,  
    const string& fileName  
);
```

13. Перевіряти коректність кожного зчитаного запису.

14. Не виводити повідомлення про успішне збереження або завантаження, якщо сталася файлова помилка.

15. Закривати файл після завершення роботи. Допускається автоматичне закриття файлу після виходу об'єкта файлового потоку з області видимості.

16. Не використовувати жорстко задане ім'я файлу всередині функцій, а передавати його через параметр.

17. Для текстового файлу застосовувати зрозумілий формат, який дає змогу однозначно відновити всі дані.

18. Якщо назви одиниць можуть містити пробіли, використовувати такий формат запису й читання, який коректно опрацьовує повні рядки.

19. Для бінарного файлу відкривати потік із режимом ios::binary.

20. Не записувати об'єкт string у бінарний файл безпосереднім копіюванням його пам'яті. Окремо записувати довжину рядка, його символи та числові поля структури.

21. У функції main() залишити організацію меню та виклики функцій збереження і завантаження.

22. Перевірити роботу програми для наявного файлу, відсутнього файлу, порожнього файлу, пошкоджених даних і повторного завантаження конфігурації.

Результатом інкременту має бути програма, яка зберігає конфігурацію конвертера у файлі, завантажує її під час наступного запуску та коректно обробляє файлові помилки.

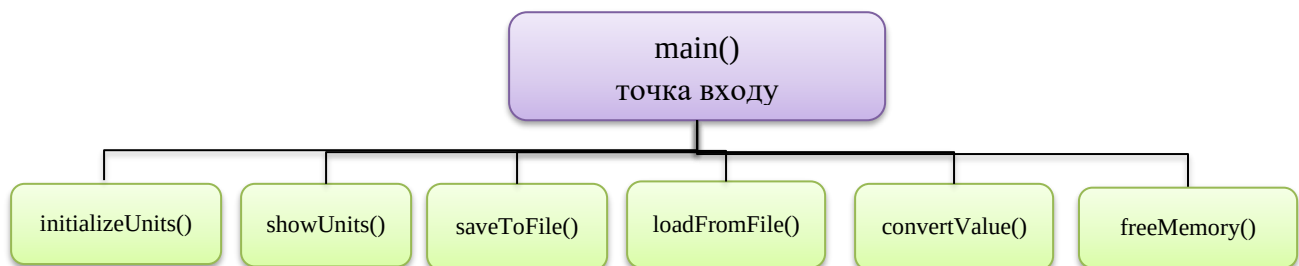


Рис. 4.21 – HIPO діаграма інкременту 11

Приклад коду та результату роботи

```
/*  
 * Проєкт: Програмна система конвертації одиниць вимірювання  
 * Інкремент: 11 – текстові та бінарні файли, фінальна інтеграція  
 *  
 * ДОДАНО В ЦЬОМУ ІНКРЕМЕНТІ:  
 * – збереження та завантаження масиву Unit у текстовому форматі;
```

```

* - збереження та завантаження масиву Unit у бінарному форматі;
* - перевірку відкриття, читання, запису, кількості записів і коефіцієнтів;
* - безпечну заміну поточного масиву лише після повного успішного читання;
* - передавання імен файлів через параметри та повернення bool.
*
* РЕФАКТОРИНГ ПОПЕРЕДНЬОГО ІНКРЕМЕНТУ:
* - дані зберігаються як цілісний масив структур Unit;
* - файлові функції не виконують консольне меню й не завершують програму;
* - повідомлення про успіх виводяться лише після успішної операції;
* - main() залишається коротким диспетчером, ресурси звільняються централізовано.
*
* НАВЧАЛЬНЕ СПРОЩЕННЯ:
* Формат файлів є локальним форматом навчального застосунку й не містить
* версії схеми або контрольної суми. Це прийнятно для поточного курсу,
* але має бути розширено у виробничій системі.
*
* ПОДАЛЬШИЙ РЕФАКТОРИНГ ПОЗА МЕЖАМИ КУРСУ:
* Доцільно перейти до std::vector, RAII, розумних покажчиків, окремих модулів
* і версіонованого формату зберігання. Ці засоби не вводяться в прикладі,
* щоб не виходити за межі тем лабораторного практикуму.
*/

```

```

#include <locale>
#include <cstdint>
#include <fstream>
#include <cctype>
#include <iomanip>
#include <iostream>
#include <limits>
#include <sstream>
#include <string>

namespace
{
    constexpr int INITIAL_UNIT_COUNT = 5;
    constexpr int MAX_DYNAMIC_VALUES = 100;
    constexpr int MAX_FILE_UNIT_COUNT = 1000;
    constexpr double CELSIUS_TO_FAHRENHEIT_SCALE = 9.0 / 5.0;
    constexpr double FAHRENHEIT_OFFSET = 32.0;

    struct Unit
    {
        std::string name;
        double toMillimeters = 0.0;
    };

    using ConversionFunction = double (*)(double);

    void clearInputStream()
    {
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    }
}

```

```

bool readInt(const char prompt[], int& value)
{
    std::cout << prompt;
    if (std::cin >> value)
    {
        return true;
    }
    std::cout << "Помилка: потрібно ввести ціле число.\n";
    clearInputStream();
    return false;
}

bool readDouble(const char prompt[], double& value)
{
    std::cout << prompt;
    if (std::cin >> value)
    {
        return true;
    }
    std::cout << "Помилка: потрібно ввести число.\n";
    clearInputStream();
    return false;
}

bool readChoice(const char prompt[], int minimum, int maximum, int& value)
{
    if (!readInt(prompt, value))
    {
        return false;
    }
    if (value < minimum || value > maximum)
    {
        std::cout << "Помилка: допустимий діапазон "
            << minimum << ".." << maximum << ".\n";
        return false;
    }
    return true;
}

bool readLine(const char prompt[], std::string& value)
{
    std::cout << prompt;
    std::getline(std::cin >> std::ws, value);
    if (value.empty())
    {
        std::cout << "Помилка: рядок не може бути порожнім.\n";
        return false;
    }
    return true;
}

bool readRepeatAnswer()

```

```

{
    char answer = 'n';
    std::cout << "Бажаєте виконати ще одну операцію? (y/n): ";
    if (!(std::cin >> answer))
    {
        clearInputStream();
        return false;
    }
    return answer == 'y' || answer == 'Y';
}

std::string normalizeName(const std::string& name)
{
    std::string normalized = name;
    for (char& symbol : normalized)
    {
        const unsigned char safeSymbol = static_cast<unsigned char>(symbol);
        symbol = static_cast<char>(std::tolower(safeSymbol));
    }
    return normalized;
}

void initializeUnits(Unit*& units, int& count)
{
    const Unit initialUnits[INITIAL_UNIT_COUNT] =
    {
        {"mm", 1.0},
        {"cm", 10.0},
        {"dm", 100.0},
        {"m", 1000.0},
        {"km", 1000000.0}
    };

    count = INITIAL_UNIT_COUNT;
    units = new Unit[count];
    for (int index = 0; index < count; ++index)
    {
        units[index] = initialUnits[index];
    }
}

void freeUnits(Unit*& units, int& count)
{
    delete[] units;
    units = nullptr;
    count = 0;
}

int findUnit(const std::string& name, const Unit units[], int count)
{
    const std::string normalizedName = normalizeName(name);
    for (int index = 0; index < count; ++index)
    {

```

```

        if (normalizeName(units[index].name) == normalizedName)
        {
            return index;
        }
    }
    return -1;
}

double convertValue(double value, const Unit& source, const Unit& target)
{
    const double valueInMillimeters = value * source.toMillimeters;
    return valueInMillimeters / target.toMillimeters;
}

void printUnits(const Unit units[], int count)
{
    std::cout << "Одиниці довжини:\n";
    for (int index = 0; index < count; ++index)
    {
        std::cout << "- " << units[index].name
                    << ": 1 " << units[index].name << " = "
                    << units[index].toMillimeters << " mm\n";
    }
}

void printConversionMatrix(const Unit units[], int count)
{
    std::cout << std::setw(10) << "";
    for (int column = 0; column < count; ++column)
    {
        std::cout << std::setw(12) << units[column].name;
    }
    std::cout << '\n';

    for (int row = 0; row < count; ++row)
    {
        std::cout << std::setw(10) << units[row].name;
        for (int column = 0; column < count; ++column)
        {
            std::cout << std::setw(12)
                        << convertValue(1.0, units[row], units[column]);
        }
        std::cout << '\n';
    }
}

std::string buildConversionReport(double value,
    const Unit& source,
    const Unit& target,
    double result)
{
    std::ostringstream report;
    report << "=====\n"

```

```

        << "ЗВІТ ПРО КОНВЕРТАЦІЮ\n"
        << "Початкове значення: " << value << ' ' << source.name << '\n'
        << "Кінцева одиниця: " << target.name << '\n'
        << "Результат: " << result << ' ' << target.name << '\n'
        << "=====\n";
    return report.str();
}

void convertByNames(const Unit units[], int count)
{
    printUnits(units, count);

    double value = 0.0;
    std::string sourceName;
    std::string targetName;
    if (!readDouble("Значення: ", value) ||
        !readLine("З якої одиниці: ", sourceName) ||
        !readLine("У яку одиницю: ", targetName))
    {
        return;
    }

    const int sourceIndex = findUnit(sourceName, units, count);
    const int targetIndex = findUnit(targetName, units, count);
    if (sourceIndex < 0 || targetIndex < 0)
    {
        std::cout << "Помилка: одну або обидві одиниці не знайдено.\n";
        return;
    }

    const double result = convertValue(value,
        units[sourceIndex],
        units[targetIndex]);
    std::cout << buildConversionReport(value,
        units[sourceIndex],
        units[targetIndex],
        result);
}

void addUnit(Unit*& units, int& count)
{
    Unit newUnit;
    if (!readLine("Назва нової одиниці: ", newUnit.name))
    {
        return;
    }
    if (findUnit(newUnit.name, units, count) >= 0)
    {
        std::cout << "Помилка: одиниця з такою назвою вже існує.\n";
        return;
    }
    if (!readDouble("Кількість міліметрів в одній новій одиниці: ",
        newUnit.toMillimeters) ||

```

```

        newUnit.toMillimeters <= 0.0)
    {
        std::cout << "Помилка: коефіцієнт має бути додатним.\n";
        return;
    }

    Unit* expandedUnits = new Unit[count + 1];
    for (int index = 0; index < count; ++index)
    {
        expandedUnits[index] = units[index];
    }
    expandedUnits[count] = newUnit;

    delete[] units;
    units = expandedUnits;
    ++count;
    std::cout << "Одиницю додано.\n";
}

void processDynamicValueArray(const Unit& source, const Unit& target)
{
    int count = 0;
    if (!readChoice("Кількість значень (1..100): ", 1, MAX_DYNAMIC_VALUES,
count))
    {
        return;
    }

    double* values = new double[count];
    for (int index = 0; index < count; ++index)
    {
        std::cout << "Значення [" << index << "]: ";
        while (!(std::cin >> values[index]))
        {
            std::cout << "Помилка. Повторіть: ";
            clearInputStream();
        }
    }

    for (int index = 0; index < count; ++index)
    {
        std::cout << values[index] << ' ' << source.name
            << " = " << convertValue(values[index], source, target)
            << ' ' << target.name << '\n';
    }
    delete[] values;
}

void runArrayConversion(const Unit units[], int count)
{
    std::string sourceName;
    std::string targetName;
    if (!readLine("Початкова одиниця: ", sourceName) ||

```

```

        !readLine("Кінцева одиниця: ", targetName))
    {
        return;
    }

    const int sourceIndex = findUnit(sourceName, units, count);
    const int targetIndex = findUnit(targetName, units, count);
    if (sourceIndex < 0 || targetIndex < 0)
    {
        std::cout << "Одиницю не знайдено.\n";
        return;
    }

    processDynamicValueArray(units[sourceIndex], units[targetIndex]);
}

double metersToKilometers(double value)
{
    return value / 1000.0;
}

double kilometersToMeters(double value)
{
    return value * 1000.0;
}

double celsiusToFahrenheit(double value)
{
    return value * CELSIUS_TO_FAHRENHEIT_SCALE + FAHRENHEIT_OFFSET;
}

double fahrenheitToCelsius(double value)
{
    return (value - FAHRENHEIT_OFFSET) / CELSIUS_TO_FAHRENHEIT_SCALE;
}

void runFunctionPointerDemo()
{
    const char* operationNames[] =
    {
        "Метри -> кілометри",
        "Кілометри -> метри",
        "Цельсій -> Фаренгейт",
        "Фаренгейт -> Цельсій"
    };
    const char* resultUnits[] = { "км", "м", "°F", "°C" };
    ConversionFunction operations[] =
    {
        metersToKilometers,
        kilometersToMeters,
        celsiusToFahrenheit,
        fahrenheitToCelsius
    };
};

```

```

constexpr int OPERATION_COUNT = 4;

for (int index = 0; index < OPERATION_COUNT; ++index)
{
    std::cout << index + 1 << " - " << operationNames[index] << '\n';
}

int choice = 0;
double value = 0.0;
if (!readChoice("Операція: ", 1, OPERATION_COUNT, choice) ||
    !readDouble("Значення: ", value))
{
    return;
}

const int index = choice - 1;
std::cout << "Результат: " << operations[index](value)
    << ' ' << resultUnits[index] << '\n';
}

// === ДОДАНО В ІНКРЕМЕНТИ 11: РОБОТА З ФАЙЛАМИ ===
bool readFileName(const char prompt[], const char defaultName[], std::string&
fileName)
{
    std::cout << prompt << " [" << defaultName << "]: ";
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    std::getline(std::cin, fileName);
    if (fileName.empty())
    {
        fileName = defaultName;
    }
    return true;
}

bool saveTextFile(const Unit units[], int count, const std::string& fileName)
{
    std::ofstream file(fileName);
    if (!file)
    {
        return false;
    }

    file << count << '\n';
    for (int index = 0; index < count; ++index)
    {
        file << std::quoted(units[index].name) << ' '
            << units[index].toMillimeters << '\n';
    }

    return static_cast<bool>(file);
}

```

```

bool loadTextFile(Unit*& units, int& count, const std::string& fileName)
{
    std::ifstream file(fileName);
    if (!file)
    {
        return false;
    }

    int loadedCount = 0;
    if (!(file >> loadedCount) ||
        loadedCount < 1 ||
        loadedCount > MAX_FILE_UNIT_COUNT)
    {
        return false;
    }

    Unit* loadedUnits = new Unit[loadedCount];
    for (int index = 0; index < loadedCount; ++index)
    {
        if (!(file >> std::quoted(loadedUnits[index].name)
            >> loadedUnits[index].toMillimeters) ||
            loadedUnits[index].name.empty() ||
            loadedUnits[index].toMillimeters <= 0.0)
        {
            delete[] loadedUnits;
            return false;
        }
    }

    delete[] units;
    units = loadedUnits;
    count = loadedCount;
    return true;
}

bool writeStringBinary(std::ofstream& file, const std::string& value)
{
    if (value.size() > std::numeric_limits<std::uint32_t>::max())
    {
        return false;
    }

    const auto length = static_cast<std::uint32_t>(value.size());
    file.write(reinterpret_cast<const char*>(&length), sizeof(length));
    file.write(value.data(), static_cast<std::streamsize>(length));
    return static_cast<bool>(file);
}

bool readStringBinary(std::ifstream& file, std::string& value)
{
    std::uint32_t length = 0;
    file.read(reinterpret_cast<char*>(&length), sizeof(length));
    if (!file || length > 10240)

```

```

    {
        return false;
    }

    value.resize(length);
    if (length > 0)
    {
        file.read(&value[0], static_cast<std::streamsize>(length));
    }
    return static_cast<bool>(file);
}

bool saveBinaryFile(const Unit units[], int count, const std::string& fileName)
{
    std::ofstream file(fileName, std::ios::binary);
    if (!file)
    {
        return false;
    }

    file.write(reinterpret_cast<const char*>(&count), sizeof(count));
    for (int index = 0; index < count; ++index)
    {
        if (!writeStringBinary(file, units[index].name))
        {
            return false;
        }
        file.write(reinterpret_cast<const char*>(&units[index].toMillimeters),
            sizeof(units[index].toMillimeters));
        if (!file)
        {
            return false;
        }
    }
    return true;
}

bool loadBinaryFile(Unit& units, int& count, const std::string& fileName)
{
    std::ifstream file(fileName, std::ios::binary);
    if (!file)
    {
        return false;
    }

    int loadedCount = 0;
    file.read(reinterpret_cast<char*>(&loadedCount), sizeof(loadedCount));
    if (!file || loadedCount < 1 || loadedCount > MAX_FILE_UNIT_COUNT)
    {
        return false;
    }

    Unit* loadedUnits = new Unit[loadedCount];

```

```

for (int index = 0; index < loadedCount; ++index)
{
    if (!readStringBinary(file, loadedUnits[index].name))
    {
        delete[] loadedUnits;
        return false;
    }
    file.read(reinterpret_cast<char*>(&loadedUnits[index].toMillimeters),
        sizeof(loadedUnits[index].toMillimeters));
    if (!file || loadedUnits[index].name.empty() ||
        loadedUnits[index].toMillimeters <= 0.0)
    {
        delete[] loadedUnits;
        return false;
    }
}

delete[] units;
units = loadedUnits;
count = loadedCount;
return true;
}

void reportFileOperation(bool success, const char successMessage[])
{
    std::cout << (success ? successMessage : "Помилка файлової операції.") <<
'\n';
}

void runMenu(Unit*& units, int& count)
{
    std::cout << "=====\n";
    std::cout << "  КОНВЕРТЕР ОДИНИЦЬ ВИМІРЮВАННЯ\n";
    std::cout << "=====\n";
    std::cout << "1 - Конвертація за назвами\n";
    std::cout << "2 - Додати одиницю\n";
    std::cout << "3 - Показати одиниці\n";
    std::cout << "4 - Показати матрицю коефіцієнтів\n";
    std::cout << "5 - Обробити масив значень\n";
    std::cout << "6 - Базова конвертація через покажчик на функцію\n";
    std::cout << "7 - Зберегти у текстовий файл\n";
    std::cout << "8 - Завантажити з текстового файлу\n";
    std::cout << "9 - Зберегти у бінарний файл\n";
    std::cout << "10 - Завантажити з бінарного файлу\n";

    int choice = 0;
    if (!readChoice("Ваш вибір: ", 1, 10, choice))
    {
        return;
    }

    switch (choice)
    {

```

```

case 1:
    convertByNames(units, count);
    break;
case 2:
    addUnit(units, count);
    break;
case 3:
    printUnits(units, count);
    break;
case 4:
    printConversionMatrix(units, count);
    break;
case 5:
    runArrayConversion(units, count);
    break;
case 6:
    runFunctionPointerDemo();
    break;
case 7:
{
    std::string fileName;
    readFileName("Ім'я текстового файлу", "units.txt", fileName);
    reportFileOperation(saveTextFile(units, count, fileName),
        "Дані збережено у текстовий файл.");
    break;
}
case 8:
{
    std::string fileName;
    readFileName("Ім'я текстового файлу", "units.txt", fileName);
    reportFileOperation(loadTextFile(units, count, fileName),
        "Дані завантажено з текстового файлу.");
    break;
}
case 9:
{
    std::string fileName;
    readFileName("Ім'я бінарного файлу", "units.bin", fileName);
    reportFileOperation(saveBinaryFile(units, count, fileName),
        "Дані збережено у бінарний файл.");
    break;
}
case 10:
{
    std::string fileName;
    readFileName("Ім'я бінарного файлу", "units.bin", fileName);
    reportFileOperation(loadBinaryFile(units, count, fileName),
        "Дані завантажено з бінарного файлу.");
    break;
}
}

```

```

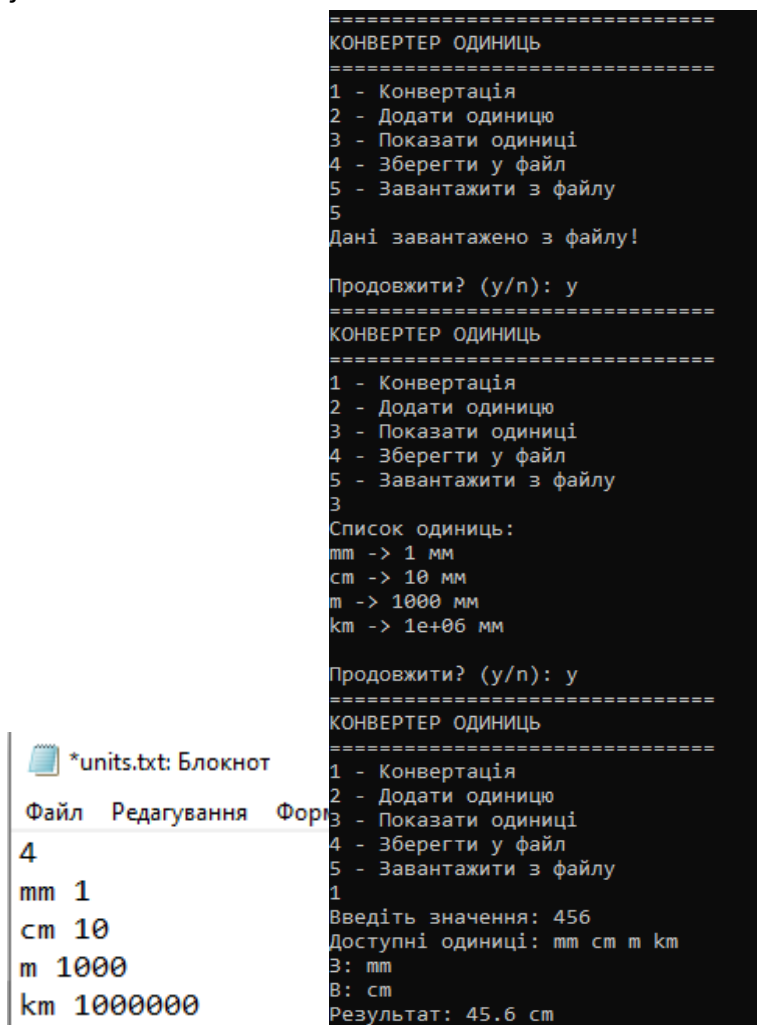
int runApplication()
{
    Unit* units = nullptr;
    int unitCount = 0;
    initializeUnits(units, unitCount);

    do
    {
        runMenu(units, unitCount);
        std::cout << '\n';
    } while (readRepeatAnswer());

    freeUnits(units, unitCount);
    return 0;
}
} // namespace

int main()
{
    std::setlocale(LC_ALL, "");
    return runApplication();
}

```



```

=====
КОНВЕРТЕР ОДИНИЦЬ
=====
1 - Конвертація
2 - Додати одиницю
3 - Показати одиниці
4 - Зберегти у файл
5 - Завантажити з файлу
5
Дані завантажено з файлу!

Продовжити? (y/n): y
=====
КОНВЕРТЕР ОДИНИЦЬ
=====
1 - Конвертація
2 - Додати одиницю
3 - Показати одиниці
4 - Зберегти у файл
5 - Завантажити з файлу
3
Список одиниць:
mm -> 1 мм
cm -> 10 мм
m -> 1000 мм
km -> 1e+06 мм

Продовжити? (y/n): y
=====
КОНВЕРТЕР ОДИНИЦЬ
=====
1 - Конвертація
2 - Додати одиницю
3 - Показати одиниці
4 - Зберегти у файл
5 - Завантажити з файлу
4
Введіть значення: 456
Доступні одиниці: mm cm m km
З: mm
В: cm
Результат: 45.6 cm

```

Рис. 4.22 Приклад роботи інкременту 11

4.11.4 Контрольні запитання

1. Чим відрізняються бінарні файли від текстових?
2. Чим відрізняється послідовний доступ до компонентів файла від прямого доступу?
3. У чому полягає відмінність між обробкою текстових файлів посимвольно від обробки по рядках?
4. Як зв'язати логічний файл із фізичним?
5. Які способи зчитування та запису текстових файлів надає мова С?
6. У чому принципова різниця між `FILE*` (С) та потоками `ifstream/ofstream` (С++)?
7. Що таке буферизація потоку і як вона впливає на запис у файл?
8. Чи можна одночасно читати і писати в один файл через `fstream`? Які умови?
9. Що потрібно зробити, щоб повторно читати файл після `eof()` у С++?
10. Чому потоки С++ вважаються більш «безпечними» ніж функції в С?

РОЗДІЛ 5 ВИМОГИ ДО ПРОГРАМНОГО КОДУ, ЗАХИСТУ ІНКРЕМЕНТІВ І ЗВІТУ (ЛАБОРАТОРНИХ РОБІТ)

5.1. Вимоги до програмного коду

Якість програмного коду є невід'ємною складовою оцінювання лабораторних робіт. Студент зобов'язаний дотримуватися вимог до структуризації програми, стандартів кодування та коментування, що відповідають професійній культурі розробки програмного забезпечення мовами C/C++.

5.1.1 Структуризація програми. Принципи процедурного та структурного програмування

Процедурно-орієнтоване програмування — це парадигма, в основі якої лежить декомпозиція програми на процедури (функції): відносно незалежні іменовані блоки коду, кожен з яких виконує одну логічно завершену дію. Структурне програмування розвиває цю ідею, обмежуючи керуючі конструкції трьома базовими: послідовністю, розгалуженням і циклом.

Застосування цих парадигм у навчальному проєкті означає, що програма будується за принципом *«розділай і володарюй»*: велика задача декомпонується на менші підзадачі, кожна з яких реалізується окремою функцією. Розробка ведеться методом *«зверху вниз»* — від загальної структури програми до деталей реалізації кожної функції.

Основні принципи структуризації програми:

Єдина відповідальність (SRP). Кожна функція розв'язує одну і лише одну локальну задачу. Не допускається поєднання в одній функції введення даних, обчислень і виведення результатів — це три різні задачі, кожна з яких реалізується окремо.

Функція `main()` як диспетчер. Функція `main()` є точкою входу до програми та коротким описом її загальної логіки. Вона викликає інші функції, але не містить обчислювальної логіки безпосередньо. Читаючи `main()`, користувач повинен одразу розуміти загальну структуру програми.

Розмір функцій. Функція не повинна перевищувати 15-25 рядків коду. Якщо функція розростається — це сигнал до подальшої декомпозиції.

Мінімізація глобальних змінних. Дані передаються між функціями через параметри та значення, що повертаються. Використання глобальних змінних зводиться до мінімуму і допускається лише у виправданих випадках із обов'язковим коментарем.

Уникнення глибокої вкладеності. Рівень вкладеності керуючих конструкцій не повинен перевищувати 3-4. Глибока вкладеність ускладнює читання, тестування та супровід коду.

Повторне використання коду. Фрагменти коду, що повторюються в різних місцях програми, виносяться у окрему функцію і викликаються за потреби. Дублювання логіки є грубим порушенням принципу DRY.

Структура файлу програми повинна відповідати такому порядку:

- коментар-заголовок файлу (назва проєкту, номер інкременту, автор, група, дата).

- директиви препроцесора (`#include`, `#define`).
- оголошення констант та типів даних (`struct`, `typedef`).
- прототипи функцій.
- функція `main()`.
- визначення функцій у логічній послідовності.

5.1.2 Дотримання стандартів кодування

Код програми має бути написаний у єдиному стилі впродовж усього проекту. Стил кодування – це не естетична перевага, а професійна необхідність: код читають інші люди, і його зрозумілість напряду впливає на якість та швидкість розробки.

Іменування ідентифікаторів

Імена змінних, функцій і констант мають бути мнемонічними – відображати зміст і призначення об'єкта. Забороняється використання беззмістовних імен (`a`, `b`, `x1`, `tmp2`) за межами лічильників циклів, а також «магічних чисел» безпосередньо у виразах.

Рекомендовані угоди про іменування наведено у таблиці 5.1.

Таблиця 5.1 – Угоди про іменування ідентифікаторів

Об'єкт	Стил	Приклад
Змінна	camelCase	studentCount, totalSum
Функція	camelCase	calculateAverage(), printReport()
Константа	UPPER_SNAKE_CASE	MAX_SIZE, PI
Структура	PascalCase	Student, FlightRecord

Форматування коду

При написанні коду дотримуються таких правил форматування:

- відступи виконуються за допомогою 4 пробілів або 1 табуляції – єдиний стил витримується у межах усього файлу;
- фігурні дужки `{}` розміщуються на окремих рядках під ключовим словом (стил Allman), що відповідає налаштуванням редактора VS Code;
- між логічними блоками коду залишається один порожній рядок;
- довжина рядка не перевищує 100 символів;
- після ключових слів `if`, `for`, `while`, `switch` перед дужкою ставиться пробіл; між іменем функції та дужкою виклику пробіл не ставиться.

Принципи чистого коду

При написанні програми студент керується принципами, описаними у розділі 2 цих методичних вказівок:

- KISS – реалізація має бути якомога простішою; складність вноситься лише тоді, коли вона є необхідною;
- DRY – логіка, що повторюється, виноситься у функцію; дублювання коду є неприпустимим;
- YAGNI – реалізується лише та функціональність, яку вимагає поточний інкремент;

- Fail Fast – коректність вхідних даних перевіряється на початку функції; програма не продовжує роботу з хибними даними;
- Defensive Programming – функції коректно обробляють граничні та некоректні значення вхідних параметрів.

Заборонені прийоми

У програмному коді не допускається:

- використання неініціалізованих змінних;
- застосування оператора `goto`;
- дублювання фрагментів коду замість виклику функції;
- використання «магічних чисел» без оголошення іменованої константи;
- надмірне використання глобальних змінних для передачі даних між функціями.

5.1.3 Коментування програмного коду

Коментарі є обов'язковою складовою програмного коду і оцінюються нарівні з його функціональністю. Грамотно написані коментарі відповідають на три запитання: «**Що?**» робить цей код, «**Як?**» це реалізовано і «**Чому?**» обрано саме такий підхід. Коментар не повинен дублювати очевидний зміст оператора — він має додавати змістовну інформацію, яку не можна прочитати безпосередньо з коду.

Розрізняють два типи коментарів.

Документаційні коментарі розміщуються перед кожною функцією та описують її призначення, вхідні параметри і результат, що повертається. Оформлюються у форматі блокового коментаря, початкові символи `///` з символами :

Пояснювальні коментарі розміщуються всередині функцій перед логічно значущими блоками коду та пояснюють суть алгоритму або нетривіального рішення. Оформлюються як однорядкові коментарі `//`:

Обов'язковому коментуванню підлягають:

- *заголовок файлу*: назва проєкту, номер інкременту, прізвище та ім'я студента, номер групи, дата виконання;
- *кожна функція*: документаційний коментар із описом призначення, параметрів і результату;
- *оголошення масивів, структур і глобальних змінних* із зазначенням їх ролі у програмі;
- *нетривіальні алгоритми*, складні умовні конструкції та цикли з неочевидною логікою завершення;
- місця, де прийнято нестандартне або компромісне рішення із зазначенням причини.

Коментарі пишуться українською мовою, грамотно і стисло. Коментування кожного рядка коду є неприпустимим: надлишкові коментарі захаращують програму і знижують її читабельність. Якщо рядок коду потребує коментаря через свою очевидність — це свідчення того, що проблема не в коментарі, а в якості самого коду.

5.2 Вимоги до звітної документації

Підготовка технічної документації на систему, програму, пристрій є важливою частиною всього комплексу робіт, які виконує розробник. Студент повинен вміти складати технічну документацію на програмне забезпечення, яке він розробляє. Тому звітам з лабораторних робіт з дисципліни «Основи програмування» приділяється особлива увага викладача. Пропонується виконувати звіти у формі гіпертекстового документа з гіперпосиланнями на лабораторні роботи, пункти вмісту звіту, файли, які демонструють роботу програм. Для створення звіту рекомендовано використовувати безкоштовний конструктор сайтів Google Sites (<https://apix-drive.com/ua/blog/reviews/schto-take-google-sites>). Форма звіту узгоджується з викладачем.

5.2.1 Вміст звіту

Поданий зміст звіту є не випадковим переліком, а логічно вибудованою структурою, що відповідає як інженерному циклу розробки програмного продукту, так і дидактичним принципам формування професійних компетентностей.

Структура звіту відображає повний цикл розробки ПЗ:

постановка задачі → аналіз → моделювання → алгоритмізація → реалізація → тестування → валідація → рефлексія.

Саме така послідовність відповідає як інженерним стандартам розробки ПЗ, так і сучасним освітнім підходам (competency-based learning).

1. Мета роботи. Фіксує дидактичну ціль лабораторної: які знання, вміння та навички мають бути сформовані. Обґрунтовує вектор розвитку поточного інкременту. Студент має чітко розуміти, яку нову функціональність він додає до свого проєкту (наприклад, перехід від лінійних обчислень до розгалужених) і які нові компетенції при цьому опановує. Це забезпечує узгодженість між завданням і результатом навчання (learning outcomes) та дає критерій для подальшої оцінки.

2. Умова задачі (інкременту). Це "Технічне завдання" (ТЗ) конкретного етапу розробки. Чітке формулювання задачі задає формальну постановку проблеми: вхідні дані, обмеження, очікувані результати. Це необхідно для коректного переходу до математичної моделі й алгоритмізації. У контексті інкрементального підходу тут важливо вказати не лише нове завдання, а й те, як воно інтегрується в уже існуючу систему (наприклад, «Додати до системи керування БПЛА модуль розрахунку залишку палива»).

3. Аналіз задачі, теоретичні обґрунтування вибраного методу вирішення задачі, розрахункові формули, посилання на літературні джерела.

Це ключовий науково-методичний блок, найважливіший етап інженерної підготовки, який демонструє розуміння суті задачі:

Чому це важливо? Програміст повинен спочатку розв'язати задачу «на папері». Це дозволяє відокремити математичні чи логічні помилки від синтаксичних помилок у коді.

Що це дає студенту?

– обґрунтований вибір алгоритмів, а не просто їх використання формує свідомість студента та оптимізує його мислення в процесі розробки коду. Студент перестає бути «кодувальником», який копіює готові рішення, і стає архітектором. Він вчиться розуміти, чому один алгоритм кращий за інший для конкретних умов (наприклад, за швидкістю чи використанням пам'яті). Він розвиває здатність оцінювати складність рішень. Це вчить

усвідомлено проєктувати рішення, розуміючи обмеження та можливості обраних методів і закладає фундамент для написання високоефективного коду;

- опис математичних моделей (формул) розвивають логічне мислення, що дозволяє студенту усвідомити фізичну чи математичну суть задачі. Якщо студент розуміє формулу, він зможе реалізувати її будь-якою мовою програмування, оскільки код — це лише інструмент реалізації. Також наявність математичного опису дозволяє провести верифікацію алгоритму ще до написання коду. Це привчає студента до точності та відповідальності, оскільки помилка у формулі на папері значно дешевша за помилку в програмній системі, що вже запущена;

- посилення на використанні джерела (підручники, наукові статті, Інтернет джерела, інструменти штучного інтелекту) формує навичку поважати інтелектуальну власність та визнавати внесок інших дослідників, що є ознакою академічної доброчесності та професійної культури. Також студент вчиться знаходити надійні джерела, працювати з документацією та фаховою літературою. Це критично важлива навичка для ІТ-галузі, де технології оновлюються щороку, і здатність швидко знайти та опрацювати першоджерело є ознакою високої кваліфікації.

4. Блок-схема алгоритму програми. Це візуалізація логіки інкременту. Блок-схема алгоритму забезпечує структурне мислення; дозволяє перевірити коректність логіки до програмної реалізації; відповідає принципу поетапної деталізації (top-down design) розв'язку задачі. Блок-схема дозволяє відокремити алгоритмічну помилку від синтаксичної. Це критично при розробці складних циклів або рекурсивних функцій (як у розділі 5), де візуальне представлення допомагає уникнути логічних зациклень. Розробка блок-схеми – це проміжний рівень між абстрактним, формальним описом способу розв'язання задачі, який не прив'язаний до мови програмування, та кодом (програмою), що є конкретною реалізацією алгоритму на певній мові програмування (C++).

5. Текст програми (скріншоти заборонені!!!) з коментарями та специфікаціями. Цей пункт демонструє володіння професійною культурою програмування. Фактично це демонстрація реалізації принципів clean code та self-documented code. Відмова від скріншотів акцентує увагу на доступності коду для копіювання, пошуку та автоматизованої перевірки (наявність плагіату, статичний аналіз). Наявність коментарів пояснює логіку програми. Оскільки проєкт триває весь семестр, без опису функцій, їх параметрів, локальних і глобальних змінних студент сам заплутається у своєму коді через кілька тижнів. Це розвиває навичку створення програм, які легко підтримувати (Maintainability).

6. Результати виконання роботи (екранні копії). Це візуальне підтвердження того, що програма не просто написана, а успішно скомпільована та запущена. Скріншоти консолі з різними вхідними даними демонструють працездатність інтерфейсу та коректність відображення результатів. Це емпіричний доказ правильності програмної реалізації алгоритму.

7. Аналіз достовірності результатів. Це критичний етап верифікації (тестування) не лише коду, а й усієї логічної цілісності проєкту. Один із найважливіших пунктів звіту. В результаті верифікації результатів роботи програми студент перевіряє розрахунки альтернативними методами, зокрема на калькуляторах, або здійснює аналітичний розв'язок математичної або фізичної задачі. Вміння аналізувати достовірність результатів роботи програми формує критичне мислення. Студент переконується, що програма веде себе передбачувано при різних вхідних даних: як при звичайних значеннях, так і при граничних (наприклад, введення нуля, від'ємних чисел або дуже великих значень). В контексті інкрементального підходу оскільки кожна нова лабораторна додає код до вже існуючого, важливо перевірити, чи не «зламала» нова функція те, що працювало раніше. Це вчить студента цілісному погляду на проєкт. Наприклад, додавши рекурсивний пошук (інкремент 5), студент має переконатися, що звичайне введення даних (інкремент 1) все ще працює коректно.

8. Висновки. Це рефлексія (механізм адаптації системи через аналіз кожного інкременту) та підсумок інкременту. В інкрементальній моделі програмний продукт створюється по частинах (інкрементах). Після кожного інкременту виконується рефлексія, яка відповідає на три ключові питання:

Що було зроблено? Аналіз досягнення мети, поставленої в п. 1 звіту.

Наскільки це коректно і ефективно? Описана якість програмної реалізації. Також дається чесна оцінка труднощів. Це важливо для планування наступного інкременту — якщо певна функція не була реалізована зараз, вона стає "технічним боргом", який потрібно закрити в наступній лабораторній роботі.

Що потрібно змінити далі? Це є адаптація як процес обґрунтованого вдосконалення системи на основі рефлексії з метою покращення наступного інкременту. В контексті інкрементального підходу це виправлення логічних помилок після перевірки; уточнення алгоритму після аналізу; покращення коду після тестування; зміна підходу, якщо результат некоректний.

Запропонована структура звіту з кожного інкременту перетворює його з простого опису дій на технічний паспорт програмного продукту, що готує студента до професійного написання технічної документації в майбутньому.

5.3 Критерії оцінювання

Оцінювання результатів виконання лабораторних робіт у межах даного курсу базується на принципах прозорості, системності та поетапного контролю якості програмного продукту. Оскільки навчання побудоване за інкрементальною моделлю, оцінка за кожну роботу є комплексною і відображає не лише факт працездатності коду, а й глибину розуміння студентом архітектурних рішень та чистоту інженерного виконання.

Критерії оцінювання спрямовані на стимулювання студента до створення надійного, структурованого та задокументованого програмного забезпечення. Особлива увага приділяється здатності студента інтегрувати нові функції у вже існуючий проєкт без порушення його цілісності (регресійна стійкість), а також вмінню аргументовано захищати обрані алгоритмічні рішення.

Підсумковий бал за інкремент визначається **інтегральним показником** виконання роботи, що включає її складність, технічну якість, наявність звітної документації, дотримання термінів (deadline) та результати захисту. Нижче подана рейтингова таблиця балів, які нараховуються за кожний інкремент.

5.3.1 Якість коду інкременту (оцінювання коду лабораторної роботи)

Якість програмного коду є визначальним чинником успішності проєкту, побудованого за інкрементальною моделлю. Оскільки кожна наступна лабораторна робота інтегрується в уже існуючу систему, до сирцевого коду висуваються вимоги, що виходять за межі простої працездатності алгоритмів. Оцінювання якості коду в межах інкременту спрямоване на перевірку його архітектурної чистоти, надійності та придатності до подальшого розширення. Якість коду доцільно оцінювати не лише через абстрактні критерії (читабельність, ефективність тощо), а через їх конкретизацію у вигляді інженерних принципів написання чистого коду. Якість коду – це інтегральний показник, складовими якого є (табл. 5.2):

Таблиця 5.2 – Метрики якості коду, що оцінюються при реалізації інкременту

№ п.п.	Метрика якості	Принципи clean code (eng)	Принципи чистого коду (укр)
1	Читабельність	Clean Code, Principle of Least Astonishment	Чистий код Принцип найменшого здивування
2	Структурованість	SRP, High Cohesion, Low Coupling	Єдина відповідальність Висока зв'язність Низька зв'язність
3	Підтримуваність	Boy Scout Rule, DRY	Правило бойскаута Не повторюйся
4	Простота	KISS, YAGNI	Зроби просто Вам це не знадобиться
5	Надійність	Fail Fast, Defensive Programming	Принцип негайної відмови Захисне програмування
6	Коректність	Clean Code, SRP, Defensive Programming	Чистий код Єдина відповідальність Захисне програмування
7	Ефективність	KISS, YAGNI	Зроби просто Вам це не знадобиться

Деталізація принципів програмування, реалізацію яких оцінює викладач.

1. Код легко читається, і його поведінка є передбачуваною. Імена змінних, структура та логіка не викликають неоднозначності.

2. Код має чітку архітектуру: кожен модуль виконує одну функцію; елементи всередині модуля логічно пов'язані; модулі мінімально залежать один від одного.

3. Код легко змінювати та розширювати: відсутнє дублювання; код поступово покращується; зміни не призводять до каскадних помилок.

4. Рішення не є надмірно складним: використано найпростіший достатній підхід; відсутній зайвий функціонал.

5. Код коректно реагує на помилки; перевіряє вхідні дані; не допускає «тихих» збоїв, що призводять до неправильного або некоректного результату.

6. Правильність результатів забезпечується через чітку логіку; відсутність змішаних відповідальностей; контроль вхідних даних.

7. Код не перевантажений зайвими обчисленнями або структурами, його, ефективність більше залежить від **алгоритму**, ніж від стилю коду.

5.3.2 Якість захисту (оцінювання ступеня володіння інтелектуальною власністю на програмний продукт)

Захист інкременту є обов'язковою складовою виконання лабораторної роботи і демонструє глибину розуміння студентом реалізованого програмного рішення. Оцінювання захисту здійснюється викладачем у форматі усного діалогу та практичної демонстрації роботи програми. Якість захисту — це інтегральний показник, складовими якого є:

1. *Розуміння коду* (Когнітивний рівень). Студент повинен вільно орієнтуватися у власній програмі: пояснювати призначення кожної функції, логіку алгоритму, обґрунтовувати вибір типів

даних та структуру програми. Студент повинен пояснити призначення будь-якого рядка коду. Не допускається захист коду, написаного без розуміння його змісту. Якість захисту вважається низькою, якщо студент не може пояснити логіку роботи власноруч написаних алгоритмів. Якщо студент використовував інструменти штучного інтелекту і не може пояснити код, згенерований ШІ, це свідчить про відсутність сформованого розуміння алгоритму та принципів реалізації, що унеможливорює зарахування роботи як самостійно виконаної та вимагає додаткового пояснення або доопрацювання.

2. *Аргументованість рішень.* Якість захисту проявляється у вмінні студента навести технічні аргументи на користь своєї реалізації. Студент має відповідати на теоретичні запитання, що стосуються теми поточного інкременту та пов'язаних конструкцій мови C/C++. Перелік контрольних запитань наведено в описі кожної лабораторної роботи.

3. *Демонстрація роботи програми* (Стресостійкість програмного коду). Студент запускає програму, вводить тестові дані та демонструє коректність результатів. На вимогу викладача студент вводить нові вхідні дані та пояснює отримані результати. Викладач може попросити змінити умову задачі (наприклад, змінити напрямок сортування або додати додаткову перевірку). Якість захисту визначається тим, наскільки швидко та впевнено студент орієнтується у власній архітектурі для внесення цих змін.

4. *Культура презентації та термінологія.* В процесі захисту студент має продемонструвати володіння професійною мовою, використовувати правильні терміни (наприклад, інкремент, ітерація, рекурсія, область видимості, розіменування покажчика) замість побутового опису дій. Вміння чітко та коротко доповісти про результати, виділяючи головне (що було додано і як це працює) свідчить про професійну культуру.

5. *Демонстрація верифікації.* Студент повинен продемонструвати доказовість та обґрунтованість результатів, підтверджуючи коректність роботи програми порівняльними розрахунками або результатами тестування. Це означає, що студент не просто показує результат на екрані, а свідомо демонструє роботу програми на «критичних» тестах (наприклад, некоректне введення), доводячи надійність (reliability) інкременту.

7. *Дотримання термінів захисту.* Інкремент захищається на тому занятті, яке за календарним графіком відповідає вказаному терміну (deadline). Написаний код інкременту студент зобов'язаний завантажити на Google Disk у вказаний викладачем репозиторій. Відповідно до заданого для кожного інкременту терміну в рейтингову таблицю викладач виставляє дату завантаження студентом коду. За кожний тиждень запізнення із завантаженням на Google Disk коду нараховуються штрафні бали, що зменшують загальний бал за інкремент. У випадку випередження терміну завантаженням на Google Disk коду студенту нараховуються бонусні бали, що збільшують загальний рейтинг за інкремент.

5.3.3 Якість звіту

Якісний звіт – це **технічний паспорт програмного продукту** проєкту, за яким будь-який інший програміст може зрозуміти логіку розвитку програми та переконатися в її надійності. Якість звіту — це інтегральний показник, складовими якого є (табл. 5.3):

Таблиця 5.3 – Метрики оцінювання якості звіту з виконання інкременту

№ п.п.	Назва пункту звіту	Метрика, що оцінюється	Семантика метрики
1	Мета роботи	Конкретність	Чітке визначення того, яка саме функціональність додається до проєкту в межах цього інкременту.
		Результативність	Опис того, що буде реалізовано (наприклад, «інтегрувати модуль обробки помилок у систему планування місій БПЛА»).
2	Умова задачі	Повнота	Точне формулювання завдання згідно з варіантом.
		Контекстуальність	Опис того, як це завдання співвідноситься з попереднім станом проєкту (що є вхідними даними з минулого етапу і що має стати результатом зараз).
3	Аналіз задачі та теоретичне обґрунтування	Глибина аналізу	Наявність логічного переходу від математичної моделі до алгоритму.
		Обґрунтованість	Пояснення, чому обрано саме ці типи даних, методи чи структури (наприклад, вибір рекурсії замість циклу).
		Коректність формул	Математична точність розрахункових моделей
		Академічність	Наявність посилань на першоджерела та технічну документацію, у тому числі на використані інструменти штучного інтелекту
4	Блок-схема алгоритму програми	Стандартизація	дотримання ДСТУ/ISO (використання правильних фігур для термінології, процесів, розгалужень).
		Деталізація	Відображення саме того алгоритму, який реалізований у коді.
		Читабельність	Чітка логіка переходів, відсутність перехрещень ліній, що заплутують схему.
5.	Текст програми	Чистота коду	Дотримання стандартів кодування, іменування (CamelCase або PascalCase) форматування, принципів процедурного та структурного програмування, єдиної відповідальності
		Інформативність коментарів	Наявність документаційних та пояснювальних коментарів (пояснення "чому" зроблено саме так, а не просто дублювання команд мови).
		Специфікації	Наявність опису вхідних/вихідних параметрів для кожної функції.
		Відсутність "магічних чисел":	Використання іменованих констант замість прямих числових значень.

№ п.п.	Назва пункту звіту	Метрика, що оцінюється	Семантика метрики
6.	Результати виконання роботи	Репрезентативність	Скріншоти повинні демонструвати роботу програми в різних режимах (основний сценарій, помилкове введення, граничні значення).
		Естетичність	Чіткі зображення консолі, де зрозуміло, що є введенням користувача, а що — відповіддю системи.
7.	Аналіз достовірності результатів	Доказовість	Наявність контрольного прикладу ("золотого стандарту"), розрахованого вручну або на калькуляторі.
		Порівняльність	Пряме зіставлення: «Результат програми» vs «Результат калькулятора».
		Верифікація інкременту:	Доведення того, що нові функції працюють правильно і не порушили логіку попередніх частин проєкту.
8.	Висновки	Відповідність меті	Аналіз того, чи виконано всі завдання, поставлені в пункті 1.
		Самоаналіз	Опис викликів, з якими зіткнувся студент.
		Управління розвитком	Якщо щось не вдалося реалізувати, має бути чітке пояснення причин та опис того, як цей "технічний борг" буде закрито в наступному інкременті.

5.3.4 Своєчасність виконання та подання інкрементів (Deadline)

Критерій «Своєчасність виконання та подання інкрементів» визначає рівень дотримання студентом встановлених термінів розробки та здачі програмного коду в межах кожного інкременту проєкту. Навчальна система передбачає поетапну розробку програмного продукту, де кожен інкремент має чітко визначений дедлайн, встановлений викладачем.

Реалізація цього критерію ґрунтується на дисциплінованому управлінні часом та відображає здатність студента працювати в умовах ітераційної розробки. Написаний код інкременту студент зобов'язаний завантажити на Google Drive (Google Disk) у визначений викладачем репозиторій. Фіксація факту здачі здійснюється шляхом внесення викладачем дати фактичного завантаження до рейтингової таблиці.

Оцінювання здійснюється відносно запланованого терміну подання. Передбачено систему штрафних і бонусних нарахунків, що дозволяє формалізовано враховувати як запізнення, так і випередження графіка виконання. Метрики критерію такі:

1. Коефіцієнт дотримання термінів виконання. Відображає ступінь відповідності фактичної дати здачі інкременту встановленому дедлайну. Показує, наскільки студент вчасно або із відхиленням виконав завдання, і слугує основою для коригування підсумкової оцінки.

2. Штрафна метрика запізнення. Вимірює негативний вплив прострочення терміну подання. Кожен тиждень запізнення зменшує підсумкову оцінку за інкремент у фіксованому відсотковому співвідношенні, що відображає зниження рівня дисципліни виконання.

3. Бонусна метрика випередження термінів. Визначає позитивний вплив дострокового виконання завдання. Кожен тиждень випередження дедлайну збільшує підсумкову оцінку у визначеному відсотковому розмірі, що стимулює проактивну та випереджальну роботу студента.

4. Інтегральна метрика своєчасності. Узагальнює вплив як запізнення, так і випередження термінів у єдиному показнику, який безпосередньо використовується для коригування базової оцінки інкременту.

Резюме. Набір сформованих у здобувачів вищої освіти компетентностей за результатами виконання інкрементального проєкту з основ програмування

Після завершення повного циклу лабораторного практикуму студент матиме на руках власний працюючий програмний продукт – консольний застосунок із повноцінною архітектурою, задокументованим кодом і тестовими сценаріями. Виконуючи кожен лабораторну роботу як інкремент у розвитку власного проєкту, студент набуває досвіду живої інженерної розробки, де якість, відповідальність і дотримання термінів є не вимогою викладача, а нормою професії. Але головне не сам продукт, а набір сформованих компетентностей:

Технічні компетентності – студент впевнено володіє основними конструкціями мови C++, розуміє управління пам'яттю, файловий ввід-вивід, рекурсію і структури даних не як ізольовані теми, а як взаємопов'язані інструменти єдиної системи.

Архітектурні компетентності – студент вміє декомпонувати задачу на функції, проєктувати структуру програми за принципом «зверху вниз» і нарощувати функціональність без руйнування вже існуючого коду.

Професійні компетентності – студент пише читабельний, структурований і задокументований код, складає технічну документацію, проводить тестування і публічно захищає власні рішення перед аудиторією.

Особистісні компетентності – студент набуває досвіду роботи в умовах реальних дедлайнів, вчиться планувати, рефлексувати після кожного етапу і перетворювати помилки на «технічний борг», який свідомо усувається в наступному інкременті.

Це саме той фундамент, на якому будується подальше навчання і кар'єра розробника програмного забезпечення.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Основна література

1. Ковалюк Т.В. Алгоритмізація та програмування. – Львів.: «Магнолія 2006», 2024. – 400 с.
2. Ковалюк Т.В., Петрівський В. Я. Гарний стиль програмування мовами С, С++, С#. Інструкція до написання програмного коду [Електронний ресурс]. Кафедра програмних систем і технологій / Т.В. Ковалюк, В.Я. Петрівський. – К.: КНУ імені Тараса Шевченка, 2025. – 59 с.
3. Deitel P., Deitel H. C++ How to Program. Pearson Education, 2017. – 1076p. URL: [http://103.203.175.90:81/fdScript/RootOfEBooks/E%20Book%20collection%20-%202024%20-%20B/CSE%20%20IT%20AIDS%20ML/C++ How to Program, 10th edition, Global Edition 2017 Paul Deitel.pdf](http://103.203.175.90:81/fdScript/RootOfEBooks/E%20Book%20collection%20-%202024%20-%20B/CSE%20%20IT%20AIDS%20ML/C++%20How%20to%20Program,%2010th%20edition,%20Global%20Edition%202017%20Paul%20Deitel.pdf)
4. Halterman R. Fundamentals of C++ Programming. Southern Adventist University. 2018. - 766 p.
5. International Standard ISO/IEC 9899:2023. Programming languages C.
6. Standard for Programming Language C++. ISO/IEC N4849:2020.

Додаткова література з методів та алгоритмів для проєктної роботи

1. Крєневич А. П. Алгоритми і структури даних. Підручник. – К.: ВПЦ «Київський Університет», 2021. – 200 с. URL: <https://www.mechmat.univ.kiev.ua/wp-content/uploads/2021/09/pidruchnyk-alhorytmy-i-struktury-danykh.pdf>
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein Introduction to Algorithms, Fourt Third Edition 2022. – 1312 p. URL: <https://iitdh.ac.in/~konjengbam.anand/courses/daa/CLRS4e2022.pdf>
3. Mark Allen Weiss. Data Structures and Algorithm Analysis in C++. Florida International University. 2014. – 654 p. URL: https://www.uoitc.edu.iq/images/documents/informatics-institute/Competitive_exam/DataStructures.pdf
4. Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman. The Design and Analysis of Computer Algorithms. Addison-Wesley Publishing Company, 1974 - 479 p. [http://103.83.136.203:802/KDK-%20DATA%20CENTER/2.3\)%20Knowledge%20Resources%20for%20Library%20Enrichment/E%20books/CT-IT%20Department/V%20and%20VI%20Sem/The%20Design%20and%20Analysis%20of%20Computer%20Algorithms%20\[Aho,%20Hopcroft%20&%20Ullman%201974-01-11\].pdf](http://103.83.136.203:802/KDK-%20DATA%20CENTER/2.3)%20Knowledge%20Resources%20for%20Library%20Enrichment/E%20books/CT-IT%20Department/V%20and%20VI%20Sem/The%20Design%20and%20Analysis%20of%20Computer%20Algorithms%20[Aho,%20Hopcroft%20&%20Ullman%201974-01-11].pdf)
5. Knuth D. The Art of Computer Programming. Vol. 3. Sorting and Searching. 1998. -792p. URL: <https://vivekupadhyay125.wordpress.com/wp-content/uploads/2013/08/donald-e-knuth-the-art-of-computer-programming-vol-3.pdf>