МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НТУ «ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра інформаційних технологій
та комп'ютерної інженерії

Коротенко Г.М., Коротенко Л.М., Сергєєва К.Л., Грищенко О.В., Харь А.Т.

Методичні вказівки

до виконання лабораторних робіт з дисципліни «Алгоритми та структури даних»

Для студентів факультету інформаційних технологій, що навчаються на спеціальностях
121 «Інженерія програмного забезпечення»
122 «Комп'ютерні науки»
124 «Системний аналіз»
126 «Інформаційні системи та технології»

Дніпро
2020

Коротенко Г.М., Коротенко Л.М. Сергєєва К.Л., Грищенко О.В., Харь А.Т. Методичні вказівки до виконання лабораторних робіт з дисципліни «Алгоритми і структури даних». Для студентів факультету інформаційних технологій, що навчаються на спеціальностях 121 «Інженерія програмного забезпечення», 122 «Комп'ютерні науки», 124 «Системний аналіз», 126 «Інформаційні системи та технології» / – Дніпро: НТУ «ДП», 2020. – 100 с.

Автори:
Г.М. Коротенко, д-р техн. наук
Л.М. Коротенко, канд. техн. наук
К.Л. Сергєєва, канд. техн. Наук
О.В. Грищенко, асистент
А.Т. Харь, асистент

Погоджено рішенням науково-методичної комісії спеціальності 126 «Інформаційні системи та технології» (протокол № 7 від 24.03.2020).

Методичні матеріали призначені для виконання лабораторних робіт з дисципліни «Алгоритми та структури даних» студентами спеціальностей 121 «Інженерія програмного забезпечення», 122 «Комп'ютерні науки», 124 «Системний аналіз», 126 «Інформаційні системи і технології».

Методичні вказівки включають сім лабораторних робіт. У них розглянуто практичні і методичні аспекти процесу навчання студентів дисципліні розробки структур зберігання даних і алгоритмів їх обробки, шляхом створення відповідних програмних одиниць на мові програмування С++ в повнофункціональному інтегрованому середовищі розробки (ІСР) Code::Blocks 17.12.

На снові застосування даної розробки вперше вирішено задачу міждисциплінарної (міжпредметної) або синтезованої інтеграції знань на основі підключення до освоєння студентами також і методів дисципліни конструювання компонентів програмного забезпечення у навчальному контексті взаємодії кафедр ІТКІ та ПЗКС НТУ «ДП» напряму 12 «Інформаційні технології» для формування відповідних компетенцій майбутніх фахівців.

В основу практичних робіт покладено ідею, що читання програмного коду студентами має велике значення, і йому слід навчати при освоєнні методології програмування та навичок щодо створення нових програмних розробок. Це сприяє написанню ефективного коду в складі команди у майбутньому. Також важливо використовувати структурне проектування та конструювання програм методом «зверху-вниз», принцип покрокової деталізації та базових структур обраної мови програмування для розробки, кодування та відлагодження компонентів програмного забезпечення (ПЗ) на основі корпоративних стандартів світових фірм та організацій (зокрема Google та Stanford University).

Зміст

Введення……. 4
Вимоги до написання текстів програм, що розробляються студентами в ході виконання лабораторних робіт і до оформлення звітів …..…… 9
Базова термінологія…….. 10
Лабораторна робота № 1. Базові (вбудовані) типи та структури даних мови С++. Пошук мінімальних або максимальних елементів (ключів) у статичних одновимірних і двовимірних масивах……...…….. 16
Лабораторна робота № 2. Пошук мінімальних і максимальних значень елементів (ключів) в одновимірних і двовимірних динамічних масивах…. 29
Лабораторна робота № 3. Формування і реалізація рекурсивних алгоритмів …..... 44
Лабораторна робота № 4. Методи сортування. Сортування в масивах. Основні алгоритми реалізації …….…….…… 52
4.1. Бульбашкове сортування (bubble sort)……..……..……. 54
4.2. Сортування методом вставок (insertion sort)……..…… 58
4.3. Сортування методом Шелла……..… 64
4.4. Метод швидкого сортування Хоара (quicksort)…….. 69
4.5. Сортування методом злиття (mergesort)……. 73
Лабораторна робота № 5. Методи пошуку. Пошук в масивах. Основні алгоритми реалізації….. ……..……..……..……. 79
5.1. Двійковий (бінарний) пошук ……. 80
5.2. Інтерполяційний метод пошуку …….. 82
Лабораторна робота № 6. Лінійні однозв'язні і двозв’язні списки..... 84
Лабораторна робота № 7. Структури даних: стеки, черги, дерева…….. 90
Приклад титульного аркуша……..…….. 96
Список рекомендованої літератури ……... 97
Додаткова література……. 100

Введення

У 2019 людство створило додатково до вже існуючих 1,2 трильйона гігабайт (Гб) або 1,2 * 10^{12} Гб даних, що еквівалентно 75 мільярдам плеєрів iPod ємністю 16 Гб, тобто більше десяти таких плеєрів на кожного жителя планети (включаючи дітей) .

У 2018 року кількість гіпермасштабуємих (hyperscale) дата-центрів (що містять понад мільярд серверів кожний) в світі збільшилося на 11% щодо 2017-го і досягло 430 одиниць, свідчать дані аналітичної компанії Synergy Research Group.

На початок січня 2019 року, на стадії планування або будівництва знаходилося ще 132 гіпермасштабуємих центрів обробки даних (ЦОД).

Наприклад, критичність роботи зі зростаючими обсягами даних призвела до того, що в кінці червня 2019 року уряд Австралії, в особі Агентства цифрової трансформації (DTA), уклало контракт вартістю $ 1 млрд. з компанією Amazon, в рамках якого всі федеральні і регіональні відомства, а також державні університети і держкорпорації стали на обслуговування і отримали доступ до хмарних сервісів Amazon Web Services (AWS). Також DTA уклало контракт з фірмою IBM вартістю $ 1,1 млрд. на надання різноманітних IT-послуг державним органам.

У цих умовах рішення задач накопичення нових даних і маніпулювання існуючими даними важливо як ніколи.

Поточний стан взаємодії моделей накопичення і зберігання даних, а також інструментів їх обробки можна представити наступним чином (рис. 1).

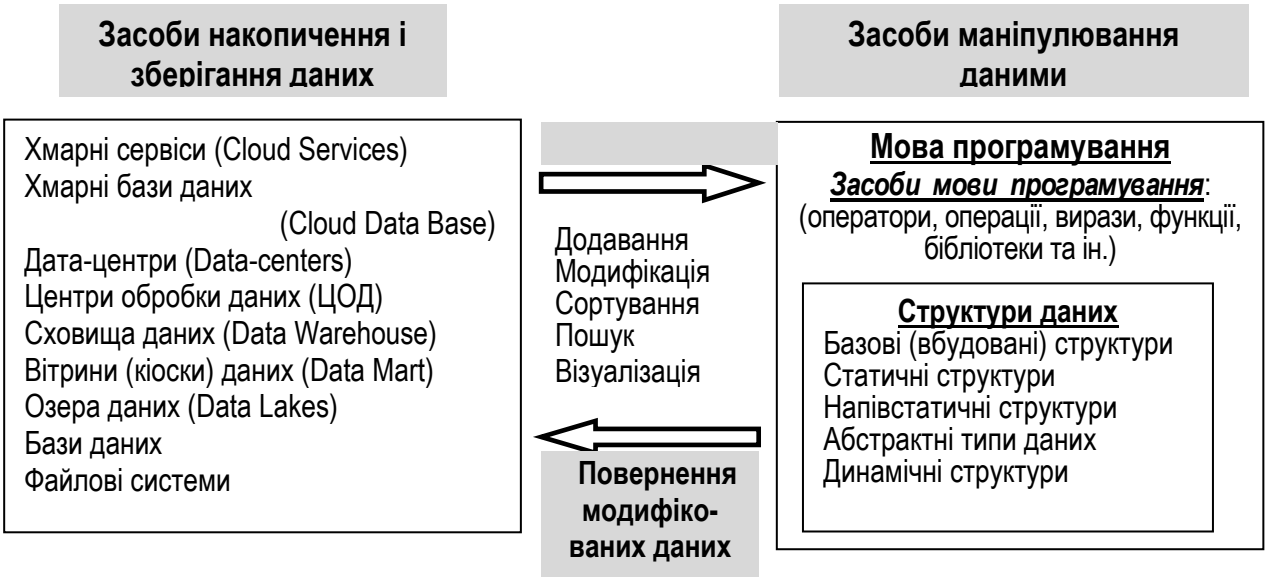

Рис. 1. Взаємодія різних видів організації зберігання даних і інструментів їх обробки

Всі структури даних (СД) в програмних системах реалізуються засобами мов програмування і є, в тому числі, у вигляді програмних одиниць (**наприклад?**).

За ознакою мінливості розрізняють наступні структури: базові (вбудовані), статичні, полустатичні, динамічні і файлові. Класифікація структур даних за ознакою мінливості на прикладі мови С ++ наведена на рис. 2.

Безліч структур даних (базових, статичних, полустатичних і динамічних) характерні для використання в оперативній пам'яті і часто називаються оперативними структурами. Файлові структури відповідають структурам даних, які розташовуються у зовнішній пам'яті.

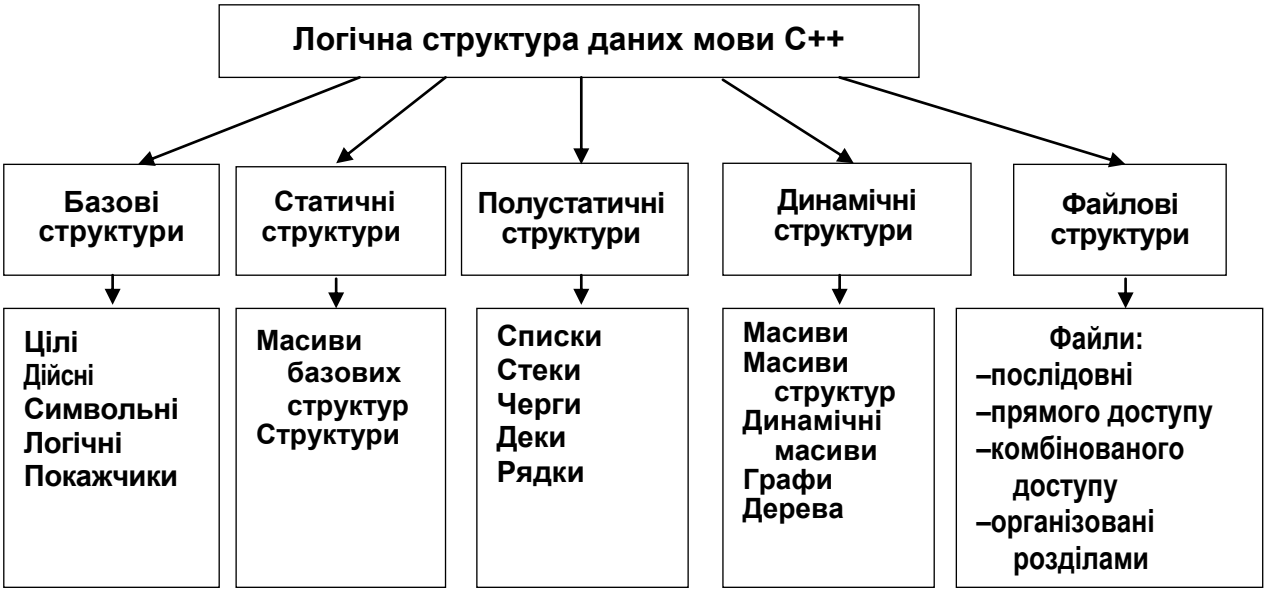

Рис. 2. Види структур даних, що реалізуються в мові С++

Складність новостворюваних і накопичуваних наборів і комплексів різноманітних даних вимагає розробки нових структур для їх зберігання і обробки. На початок 2019 року було відомі більш 15-ти видів структур зберігання даних (табл. 1).

Загальна кількість власне елементарних структур і типів перевищує 200 одиниць, для кожної з яких може бути застосована деяка кінцева кількість алгоритмів обробки. У свою чергу, реалізація кожної з відомих структур, можлива на більш ніж 50-ти найбільш популярних мовах програмування. Це робить задачу формування структур і алгоритмів їх обробки досить складною і нетривіальною.

Важливо відзначити також, і те, що в основі формування похідних структур лежать базові типи даних. Таким чином, безліч операцій обробки даних розпадаються на дві групи:

- операції роботи зі значеннями, що зберігаються в елементах структур;
- операції роботи з самими елементами структур (вершинами дерев і графів, елементами масивів і стеків, вузлами списків і черг і т.д.).

Таблиця 1

Види структур елементів зберігання даних

№ пп	Вид структури даних	Приклади структур	Заг. кільк.
1.	Базові або примитивні (Primitive)	Boolean, Character, Floating-point, Integer & etc.	**7**
2.	Складові типи або не примітивні типи (Composite types)	Array, Structure, Union	**3**
3.	Абстрактні типи даних (АТД) (Abstract data types – ADT)	Container, List (sequence), Tuple, Multimap, Set, Multiset (bag)., Stack, Queue, Double-ended queue (deque) & etc.	**10**
4.	Лінійні структури даних (Linear data structures)	Bit array, Bit field, Bitboard, Bitmap, Circular buffer, Control table, Dope vector, Hashed array tree, Iliffe vector, Hashed array tree & etc.	**19**
5.	Списки (Lists)	Doubly linked list, Array list, Linked list, Self-organizing list, Skip list & etc.	**13**
6.	Дерева. Бінарні дерева (Binary trees)	AA tree, AVL tree, Binary search tree, Cartesian tree, Left-child right-sibling binary tree, Red–black tree Tango tree & etc.	**24**
7.	Дерево пошуку (B-trees)	B-tree, B+ tree, B*-tree, B# tree (B sharp tree), Dancing tree & etc.	**11**
8.	Кучі (Купи) (Heaps)	Heap, Binary heap, B-heap, Weak heap, Fibonacci heap, Leonardo Heap, 2-3 heap & etc.	**18**
9.	Дерева (Trees)	Radix tree, Parent pointer tree, Splay tree, Suffix tree (PAT tree), Suffix array & etc.	**15**
10.	Множинні дерева (Multiway trees)	Ternary tree, K-ary tree, And–or tree, (a,b)-tree, Link/cut tree, Spaghetti stack & etc.	**14**
11.	Дерева з розбиттям простору (Space-partitioning trees)	Segment tree, Interval tree, Range tree, K-d tree, Quadtree, Relaxed k-d tree & etc.	**28**
12.	Дерева для конкретних застосувань (Application-specific trees)	Abstract syntax tree, Parse tree, Decision tree, Minimax tree, Finger tree & etc.	**10**
13.	Структуры на основі хеш-таблиць (Hash-based structures)	Bloom filter, Distributed hash table, Double hashing, Dynamic perfect hash table, Hash list, Hash tree & etc.	**16**
14.	Графи (Graphs)	Graph, Adjacency list, Adjacency matrix, Graph-structured stack, Scene graph, Multigraph, Hypergraph, Directed graph & etc.	**14**
15.	Інші структури даних	Lightmap, Winged edge, Quad-edge & etc.	**5**
		Загалом	**207**

Особливістю освоєння дисципліни «Алгоритми та структури даних» є комплексність використовуваних на різних етапах вирішення поставлених завдань видів організації засобів представлення різноманітних даних і можливих операційних і алгоритмічних можливостей конкретних мов програмування (МП).

За своєю суттю, мова програмування є посередником між конкретними структурами зберігання елементів даних і областю їх застосування (розробка Web-додатків, баз даних, інформаційних систем, систем електронної комерції і т.д.). З іншого боку, крім постійно зростаючого спектру конкретних мов програмування (JavaScript, Python, C ++, C #, Java, Ruby, Pascal, Visual Basic for Applications, Perl і т.д.), збільшується кількість:

– постійно еволюціонуючих апаратних компонентів;

– інформаційно-технологічних і цифрових платформ;

– розширення спектру різних структур даних;

– постійно оновлюваних засобів швидкої розробки додатків (Rapid Application Development - RAD)

Крім того, різні мови мають різну кількість операцій (операторів) роботи з даними (наприклад, множення, ділення, додавання, віднімання і т.д.), а також і кількість пріоритетів їх застосування (табл. 2).

Таблиця 2.

Кількість операцій (операторів) в мовах програмування і кількість їх рівней приоритетів

Найменування мови	**Кількість операторів / операцій**	**К-ть рівнів приоритету операцій**
Turbo Pascal 7.0	20	4
Visual Basic 6.0	23	4
Visual Basic for Applications	23	4
VBScript	23	9
Python	35	23
C	43	15
Ruby	43	23
Java	44	9
Java Script	48	14
C#	51	14
C++	52	18
Perl	64	17

Для того, щоб уявити різноманітність форм виведення тільки простого текстового рядка, наведемо, наприклад, організацію виведення на екран комп'ютера стандартної фрази «Привіт, світ!» (`"Hello, World!"`) на різних мовах програмування (http://helloworldcollection.de/) (табл. 3).

Таблиця 3

Приклади оформлення виводу текстового рядку "Hello, World!" в різних мовах програмування

Мова	Код програми
JavaScript	`console.log("Hello, World!");`

Pascal	`begin`
	`  WriteLn('Hello, World!');`
	`end.`

C	`#include <stdio.h>`
	`void main()`
	`{`
	`  printf("Hello, World!\n");`
	`  return 0;`
	`}`

C++	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`  cout << "Hello, World!" << endl;`
	`  return 0;`
	`}`

Java	`class HelloWorld {`
	`  static public void main( String args[] ) {`
	`  System.out.println( "Hello, World!" );`
	`  }`
	`}`

C#	`using System;`
	`namespace HelloWorld`
	`{`
	`  class Hello`
	`  {`
	`  static void Main()`
	`  {`
	`  Console.WriteLine("Hello, World!");`
	`  }`
	`  }`
	`}`

У даних методичних вказівках виконання поставлених завдань реалізується засобами мови програмування С ++.

Істотна спрямованість даної мови на роботу з апаратурою і компонентами персональних комп'ютерів забезпечується 52-ма операторами роботи з даними, об'єднаними в 18 груп різного пріоритету виконання. Слід мати на увазі, що в їх склад, зокрема, входять постфіксні і префіксні інкремент (++) і декремент (-), оператори присвоювання (=): з множенням (* =), розподілом (/ =), розподілом по модулю (% =), сумою (+ =), різницею (- =), зрушенням вліво (<< =) і зрушенням вправо (>> =). Сюди ж входять побітові операції: І (&), АБО (|), побітовое виключає АБО (^), логічне І (&&) і логічне АБО (||), а також і багато інших.

Вимоги до написання текстів програм, що розробляються студентами в ході виконання лабораторних робіт і до оформлення звітів

1. Завдання, в яких потрібно розробити програму або реалізувати заданий варіант алгоритму, необхідно виконувати на мові програмування С ++, дотримуючись принципів структурного програмування.

2. При розробці алгоритму розв'язання задачі слід домагатися реалізації його найбільшої ефективності.

3. Вихідні дані вибираються відповідно до умов завдання.

4. За погодженням з викладачем допускається спільна робота над проектом командою, що складається з двох чоловік.

5. При формуванні програмного коду користувач повинен коментувати умову задачі, вимоги до вхідних даних, а також створювати засоби для виведення на екран комп'ютера результатів роботи програмної одиниці і повідомлень, що відображають ситуації, що виникають у разі некоректного введення даних.

6. У розроблених програмах необхідно передбачити можливості роботи в одному з трьох режимів:

а) введення даних здійснюється шляхом звернення до стандартних засобів мови програмування, а виведення результатів – на консоль;

б) введення даних з клавіатури, тобто виконання відповідного алгоритму введення для заданого набору вхідних даних (введення даних і виведення результату здійснюється з використанням консолі);

в) введення даних з заздалегідь підготовленого файлу, а виведення результатів – в файл відповідного типу.

Основні вимоги до оформлення створюваного коду програмних одиниць:

1. Програма, написана на мові програмування С ++, повинна відповідати певному стилю, який визначає угоди, що додають можливість гнучко управляти створюваним кодом на мові С ++.

2. Кожна команда повинна бути представлена на окремому рядку; будь-яке оголошення властивості або змінної також має бути виконано на окремому рядку.

3. Структура програми повинна формуватися за допомогою використання необхідної кількості відступів. Необхідно використовувати тільки відступи кількістю 4 пробіла за раз для відображення структури програми для відображення структурної вкладеності мовних конструкцій. Застосування табуляції в коді заборонено.

4. Код програм необхідно супроводжувати коментарями, в кількості, достатній для швидкого розуміння алгоритму.

5. Імена ідентифікаторів лічильників циклів позначаються символами i, j, k, l, m, n.

6. Використовуйте Сі-стиль іменування змінних та функцій «`camelCase`» (з англ. – «ВерблюжійРегістр» або «ГорбатийРегістр»), де перше слово починається з малої літери, потім перша буква кожного наступного слова є прописною. на відміну від імен функцій, які повинні починатися з великих літер і мати великі літери для кожного нового вхідного слова. Наприклад, arrayName, bubbleSort.

Базова термінологія

З огляду на різноманітність термінів і визначень, що вживаються в різних лекційних курсах, літературних джерелах і навчальних посібниках, нижче наводяться смислові тлумачення найбільш уживаних формальних характеристик абстракцій програмно-мовних і алгоритмічних складових обробки даних, що застосовуються в даних методичних вказівках.

• **Дані** – зареєстровані сигнали, факти і / або їх опис.

Таким чином, всі ***дані*** характеризуються деякими **значеннями** і/або **описом цих зареєстрованих значень**.

Наприклад: 20 велосипедів, 20 танків, 20 зошитів (приклади значень даних та їх описів).

• **Значення –** чисельна характеристика змінної, що формується у обраній системі уявлення ***стандартних позначень*** (двоїчна, десятична або римська системи числення; градуси, кілометри, кілограми і т.д.).

Наприклад: 20-й кілометр, XX-е століття – Фокс (назва фірми), 20° – температура двадцять градусів, FF – адреса повернення з програми (значення адреси повернення у шістнадцятирічній системі адресації регістрів ПК, еквівалентне значенню 255 у десятичній системі числення) і т.д.

• **Елемент данних** – окреме значення набору даних.

• **Елементи групи –** елементи даних, розділених на підгрупи.

Дані поділяються на кількісні, якісні і формалізовані.

• **Кількісні дані –** дані, що вимірюються за допомогою чисел, що мають змістовний сенс (зазвичай для користувача – це дані, представлені в десятковій системі числення: 1,2, 3, 4, 5, 6, 7, 8, 9).

• **Якісні дані –** дані, представлені у формі звітів, інтерв'ю або будь-якого описового матеріалу (зазвичай для користувача – ці дані представлені у вигляді тексту).

Нижче представлені два можливих види кількісних і якісних елементів даних (рис. 3).

кількісне значення	якісне значення
числове значення	**опис**
20	**градуси**

кількісне значення	якісне значення
числове значення	**опис**
20	**книг**

Рис. 3. Приклади якісних і кількісних даних

• **Формалізація –** уявлення будь-якої змістовної області (міркувань, доказів, процедур класифікації, пошуку інформації, наукових теорій) у вигляді формальної системи або обчислення.

• **Логічні дані** – один з видів формалізації опису моделей навколишнього світу (зазвичай для користувача – значення «ЛОЖЬ», «ИСТИНА» або «TRUE», «FALSE».

• **Структура данних** – це систематичний спосіб організації даних для ефективного їх використання. Як правило, структура даних являє собою набір відповідних даних для вирішення задачі.

• **Програмна одиниця** являє собою послідовність операторів і коментарів, які описують алгоритм вирішення поставленої задачі.

• **Оператори** – основні елементи мови. Вони задають дії, які необхідно виконати для реалізації алгоритму задачі, а також описують використовувані дані і визначають структуру програмної одиниці.

Терміни, які є основоположними (фундаментальними) в області створення **структур даних**.

• **Інтерфейс**. Кожна структура даних має свій ***унікальний інтерфейс***.

• **Інтерфейс структури даних** являє собою набір операцій, які підтримуються програмної одиницею, що реалізує дану структуру даних.

• **Інтерфейс** надає тільки список підтримуваних операцій, тип параметрів, які вони можуть приймати, і повертає тип цих операцій.

• **Реалізація.** Кожна реалізація забезпечує внутрішнє представлення структури даних, а також визначення алгоритмів, використовуваних в операціях структури даних.

Як було описано вище, всі створювані структури елементів зберігання даних призначені виключно для їх подальшої обробки!

• **Обробка даних (Data processing) –** це процес збору безлічі конкретних елементів даних і **маніпулювання** ними для отримання значимої інформації.

• **Маніпулювання даними (Data manipulation)** – процес трансформації (зміни) взаємного розташування елементів, збережених у **використовуваній структурі збереження даних**, з метою їх підготовки для подальшої обробки.

Наприклад, електронний (цифровий) журнал даних про співробітників організації може бути організований в алфавітному порядку, що полегшує пошук окремих записів.

Подальша обробка даних, зазвичай зводиться до виконання одного або декількох взаємопов'язаних процесів, іменованих в такий спосіб:

– валідація;
– сортування;
– пошук;
– узагальнення;
– агрегування;
– аналіз;
– звітність;
– класифікація.

В даному документі розглядаються, в основному, процеси сортування і пошуку даних або відповідних ключів.

Термінологія етапів обробки даних у процесі сортування виглядає натупним чином.

Сортування (*Sorting*) – це будь-який процес систематичного впорядкування значень, який має два загальних, але однозначних значеннь:

– впорядкування: розміщення значень у послідовності, упорядкованій за якимсь критерієм;

– категоризація: групування значень зі схожими властивостями.

Алгоритм сортування в комп'ютингу – це алгоритм, який розміщує елементи **списку** в певній послідовності.

Найбільш часто використовуються послідовності:

– **числовий порядок;**

– лексикографічний порядок.

• **Результат роботи будь якого алгоритму сортування** (наприклад, сортування за зростанням) повинен задовольняти ***двом вимогам***.

1. Вихідні дані розташовані в неубуваючому порядку (кожен елемент не більше попереднього елемента відповідно до бажаного загального порядку).

2. Результат роботи алгоритму є перестановка (переупорядкування елементів), але зберігаються (не втрачаються) при цьому всі вихідні елементи оброблюваної безлічі елементів.

• **В компьютингу: список** або **послідовність** – це абстрактний тип даних, що представляє скінченну (рос. – *конечную*) кількість упорядкованих значень, де одне і те ж значення може зустрічатися більше одного разу.

• **Значення в структурі даних** – це представлення деякої сутності, котрою може маніпулювати програма. Члени типу є значеннями цього типу.

Наприклад, однозв'язна **структура списку**, що реалізує список з трьома цілочисельними елементами (`12, 99, 37`) може бути представлена наступним чином (рис. 4):

Рис. 4. Представлення списка, що зберігає 3 елемента

• Сортування ***списків*** (в залежності від контексту, також званого, наприклад, в мові C++ ***структурами***, що включають декілька ***полів***) може бути виконана на основе одного або декольких його компонентів, які звуться ***ключами***.

1. Ключ сортування – **компонент** або **властивість** за якими може бути відсортована послідовність даних.

2. Ключ сортування (також) – це **поле**, по значенню якого виконується сортування списків за зрастанням значень **ключа** або за зменшенням його значень.

Наприклад, у вигляді ***компонентів*** списку можуть бути представлені книги, а ***ключами сортування*** – можуть бути: а) назви книг, б) рік випуску (видання) книг, в) прізвища авторів і т.д.

Новий складений ключ сортування може бути створений з двох або більше ***ключів*** сортування у ***лексикографичному*** порядку. Перший зветься ***первинним*** ключом сортування, другий – ***вторинним*** ключом сортування і т.д.

Наприклад, ***адреси мешканців деякого населеного пункту України*** або області можуть бути відсортовані з використанням найменування міста їх проживання в якості первинного ключа сортування, а ***вулиці*** – в якості вторинного ***ключа*** сортування. Далі можуть слідувати номера: під'їздів, квартир, призвіщ мешканців квартир і т.д.

На рисунку нижче представлені різноманітні поєднання елементів реальних даних (рис. 5).

ДАНІ				
Кількісні	**Якісні**	**Якісні**	**Якісні**	**Якісні**
Значення	**Опис**	**Опис**	**Опис**	**Опис**
КОРТЕЖ				
Сутність	**Сутність**	**Сутність**	**Сутність**	**Сутність**
Поле	**Поле**	**Поле**	**Поле**	**Поле**
Ключ	**Ключ**	**Ключ**	**Ключ**	**Ключ**
20	**температура**	**градусів**	**цельсія**	**у квартирі**
20	**координата**	**градусів**	**північної**	**широти**
20	**книг**	**поезія**	**2010 р. видання**	
	Грушко	**Остап**	**інженер**	**1980 р. народження**
	адреса	**Дніпро**	**вул. Миру**	**буд. 50**

Рис. 5. Подання компонентів різноманітних даних

Найбільш поширеними структурами для зберігання і обробки даних в програмних одиницях є масиви.

Масиви характеризуються властивістю зберігати однотипні дані, але самі масиви можуть описувати різні типи збережених ними даних.

Іншими словами, масив, описаний у програмній одиниці як цілочисельний, зберігає цілі числа, речовинний – речові (дійсні), символьний – символьні і т.д.

Але один елемент масиву може зберігати тільки одне значення елемента даних (рис. 6).

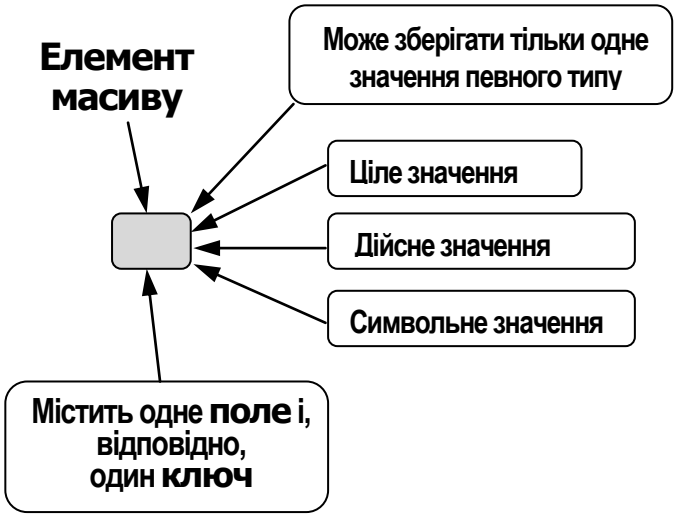

Рис. 6. Елементи даних, які мають можливість зберігання в одному елементі масиву

Для зберігання наборів різнотипних даних можуть застосовуватися т.зв. записи (в мові С – структури (`struct`), а в мові Паскаль – `records`). В даному випадку кількість полів і ключів в кожному елементі зберігання даних буде відповідати загальній кількості збережених даних (рис. 7).

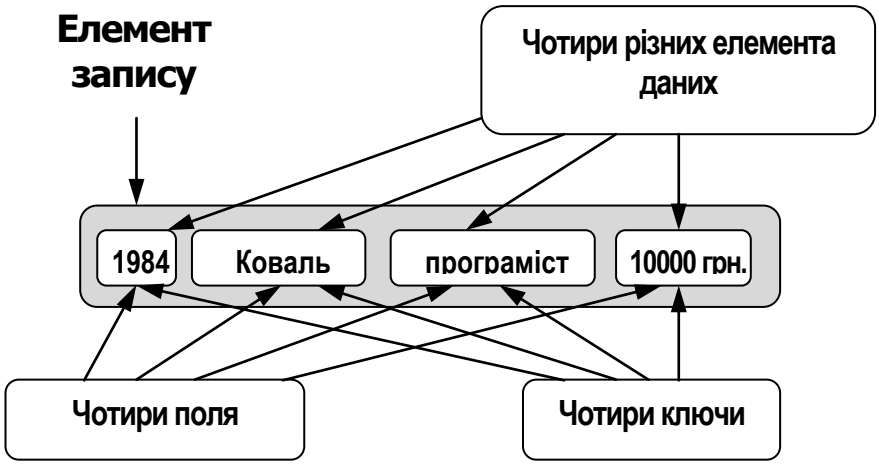

Рис. 7. Подання компонентів даних в структурах

Решта полів несуть в собі змістовну інформацію, яка витягується і використовується зі знайденого елемента зберігання даних. Таким чином, вся сукупність даних, що зберігається в одному елементі зберігання даних часто зветься полем.

Розглянемо, наприклад, набір з трьох елементів зберігання даних, що мають такі компоненти (рис. 8).

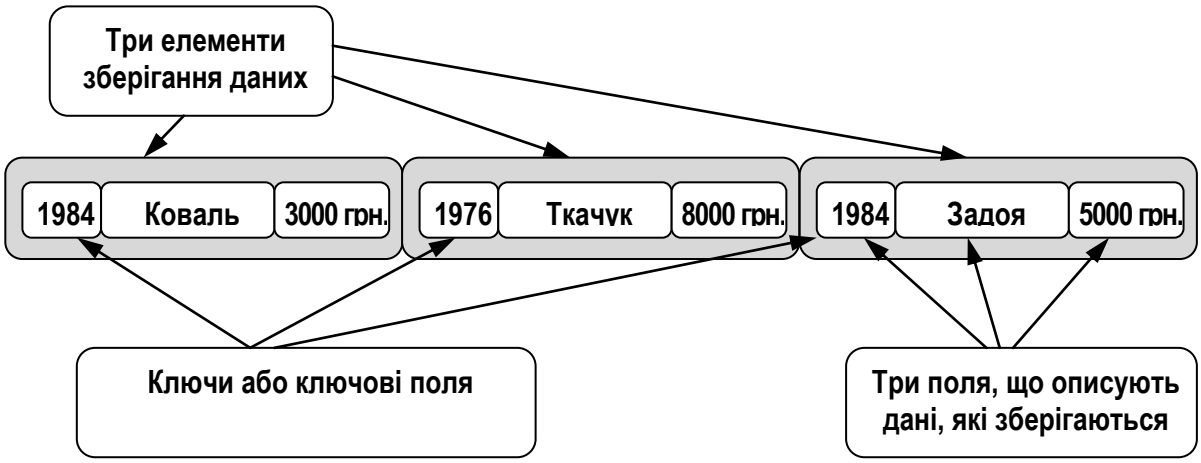

Рис. 8. Взаємозв'язок кючей, полів і елементів зберігання даних

Таким чином, в даній структурі розташовується масив наборів даних, що містять такі компоненти:

- рік народження співробітника,
- прізвище співробітника;
- зарплата співробітника.

A = {(1984, «Коваль», 3000 грн.), (1976, «Ткачук», 8000 грн.), (1984, «Задоя», 5000 грн.)}

Ці дані можна відповідно впорядкувати за трьома ключам: датами народження, прізвищами і зарплатами співробітників.

Однак, у всіх трьох випадках результуюча послідовність даних буде різна.

Більш того, якщо упорядкувати масив ***A*** за ключом «рік народження», то можна отримати два різних результату:

A^* = {(1976, «Ткачук», 8000 грн.), (1984, «Коваль», 3000 грн.), (1984, «Задоя», 5000 грн.)}

A^{**} = {(1976, «Ткачук», 8000 грн.), (1984, «Задоя», 5000 грн.), (1984, «Коваль», 3000 грн.),}

Масиви A^* и A^{**} відрізняються послідовністю розташування елементів (1984, «Коваль», 3000 грн.) і (1984, «Задоя», 5000 грн.) (хоча оба масиви упорядковані за ключом). В масиві A^* порядок елементів з однаковими ключами такий же як і в вихідному масиві A, однак в масиві A^{**} цей порядок змінений. Масив A^* отримано стабільним упорядкуванням (сортуванням), тоді як масив A^{**} отримано нестабільним упорядкуванням (сортуванням), тобто зі зміною порядку розташування елементів з однаковими ключами (тут рік народження «1984».

ЛАБОРАТОРНА РОБОТА № 1

Базові (вбудовані) типи і структури даних. Пошук мінімальних або максимальних елементів (ключів) в статичних одновимірних і двовимірних масивах.

Мета роботи: опанування принципів роботи зі статичними масивами, заповненими різнотипними значеннями елементів.

Теоретичні відомості. Статичні одновимірні масиви в С ++

Масив – це структура елементів зберігання даних, представлена у вигляді групи комірок одного, заздалегідь заданого типу, об'єднаних під одним єдиним ім'ям. Окрема комірка масиву називається елементом масиву. Як правило, вмістом комірок масиву можуть бути дані будь-якого базового типу, наявного у певній мові програмування.

Масиви можуть мати як одне, так і більш одного вимірювань.

Залежно від кількості вимірів масиви поділяються на одновимірні, двовимірні, тривимірні і так далі до ***n***-вимірного масиву.

Модель організації зберігання даних в одновимірному масиві така (рис. 1.1).

<u>Одновимірний масив з восьми елементів:</u> int a[8]

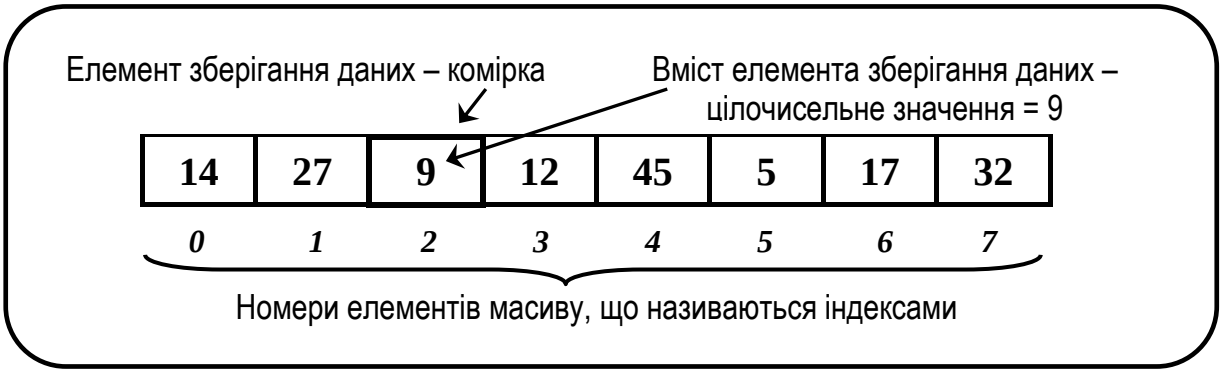

Рис. 1.1. Представлення одновимірного масиву з восьми елементів (комірок)

Синтаксис оголошення одновимірного масиву такий:

```
тип_даних, імя_масиву[кількість_елементов].
```

<u>Наприклад:</u>

```
int a [5]; // масив з 5 елементів цілого типу
float b [3];// масив з 3 елементів дійсного типу
```

❶. Можна задавати кількість елементів масиву константами:

```
const int n = 3; // специфікатор const дає можливість програмісту
                 // проінформувати компілятор, що значення даної
                 // константи не повинно змінюватися в програмі
float b [n];// масив з n дійсних елементів
```

❷. Масив можна ініціалізувати (задавати вміст елементів).

```
int a [] = {-5,12, 10, -4};// тут компілятор сам визначає
                           // кількість елементів
                           // оголошуємо масив з 4-х цілих елементів
```

❸. Можна відразу ставити число елементів масиву і його ініціалізувати:

```
float c [3] = {0.5, -6.33, 4.0}; //елементи масиву – c[0], c[1], c[2]
```

Головне – потрібно запам'ятати, що звернення до елементів масиву проводиться за допомогою індексів. Значення першого індексу завжди дорівнює нулю.

❹. Крім того, мова С ++ дозволяє заповнювати елементи масиву наборами випадкових чисел в заданому діапазоні, а також з клавіатури (див. приклад програми нижче).

Статичні двовимірні масиви в С ++

Синтаксис оголошення двовимірного масиву:

```
тип_даних ім’я_масиву [кількість_рядків] [кількість_стовпців];
```

<u>Наприклад:</u>

```
int a[2][3];//масив цілих чисел з 2-х рядків і 3-х стовпців
```

❶. Можна задавати кількість рядків і стовпців константою:

```
const int n = 2; // число рядків
const int m = 3; // число стовпців
float b[n][m];// двовимірний масив дійсних чисел
```

❷. Можна форматувати двовимірний масив (але другий індекс обов'язковий!):

```
float matr[][5] = {{3.1,5.3,5.5,7.1,8.6}, // 1-й рядок масиву
                   {4.5,2.4,3.9,5.3,6.6}, // 2-й рядок масиву
                   {7.2,8.6,9.7,7.6,3.3}}; // 3-й рядок масиву
```

в результаті отримаємо:

```
float matr[3][5]; // перший елемент – matr[0][0] = 3.1 і т.д.
```

❸. Можна одночасно: задавати число елементів масиву і форматувати значення елементів.

❹. Крім того, мова С ++ дозволяє заповнювати елементи масиву наборами випадкових чисел в заданому діапазоні, а також з клавіатури (див. приклад програми далі).

<u>Завдання до Лабораторної роботи № 1.</u>

Розробити програму для опису одновимірних і двовимірних масивів відповідно до варіантів, зазначених у табл. 1.1. і виконати такі дії.

1. Вибрати тип масиву в залежності від номера отриманого варіанту.
2. Ініціалізувати описаний масив.
3. Ввести відповідний набір випадкових величин.
4. Вивести на екран вміст одновимірного масиву (за допомогою циклу `for` і операції виведення `cout`).
5. Виконати зазначені у завданні дії і вивести результати. Зробити висновки.
6. Повторити пункти 1-5 для двовимірного масиву. Зробити висновки.

Текст програми слід доповнити повідомленнями про введення і виведення даних. Такі повідомлення полегшують контроль виконання програми. У міру виконання роботи, до відладжених фрагментів тексту необхідно додати коментарі згідно до **Вимог до виконання Завдання**.

По завершення роботи підготувати звіт і занести в звіт отримані результати. Форму титульного аркуша звіту наведено у Додатку 1.

<u>ПРИМІТКА,</u> Файл з текстом програми необхідно зберегти для виконання наступної роботи.

Таблиця 1.1.

Варіанти завдань до Лабораторної роботи № 1

№ варі-анта	№ завда-ня	Зміст завдання	Розмір-ність
1	1	Знайти суму кубів від’ємних і найменше з додатних чисел	12
	2	Знайти в масиві і вивести на екран кількість додатних і кількість від’ємних елементів масиву	5x4
2	1	Знайти середнє арифметичне окремо додатних і від’ємних чисел	15
	2	Знайти в масиві і вивести на екран середнє арифметичне від’ємних елементів кожного рядка масиву	4x6
3	1	Знайти суму абсолютних значень від’ємних елементів і кількість елементів, значення яких знаходяться в діапазоні [0..1]	11
	2	Знайти в масиві і вивести на екран суму і кількість елементів, значення яких не перевищують значення 1	5x3
4	1	Знайти максимальний і мінімальний елементи і їх порядкові номери	17
	2	Знайти в масиві і вивести на екран суму елементів кожного непарного стовпця	4x5
5	1	Знайти суму додатних елементів, а також суму і кількість від’ємних елементів	14

№ варі-анта	№ завда-ня	Зміст завдання	Розмір-ність
	2	Знайти в масиві і вивести на екран середнє арифметичне додатних елементів кожного стовпця масиву	6x4
6	1	Знайти суму елементів з парними індексами і добуток від'ємних елементів з непарними елемент	11
	2	Знайти в масиві і вивести на екран суму елементів, розташованих на головній діагоналі	4x7
7	1	Знайти добуток всіх елементів, попередньо замінивши нульові елементи одиницями	16
	2	Знайти в масиві і вивести на екран значення максимального елемента і кількість нульових елементів масиву	3x6
8	1	Знайти суми 5-ти перших елементів масиву з 10-ти елементів	10
	2	Знайти в масиві і вивести на екран суму елементів кожного парного стовпця і добуток елементів кожного непарного стовпця	5x6
9	1	Знайти добуток елементів, розташованих до першого нульового елемента	11
	2	Знайти в масиві і вивести на екран кількість і суму елементів, більших за 1	7x4
10	1	Знайти середнє арифметичне від'ємних елементів і кількість нульових елементів	13
	2	Знайти в масиві і вивести на екран середнє арифметичне всіх елементів, більших за 1	5x6
11	1	Знайти добуток абсолютних значень і кількість елементів, що знаходяться у діапазоні [-5 .. + 5]	12
	2	Знайти в масиві і вивести на екран кількість нульових елементів і замінити нульові елементи на 1	6x4
12	1	Знайти суму від'ємних елементів, кількість нульових елементів і добуток додатних елементів	19
	2	Знайти в масиві і вивести на екран суму і кількість елементів, рівних 1	5x4
13	1	Знайти суму від'ємних елементів з парними індексами і добуток всіх елементів з непарними індексами, попередньо замінивши нульові елементи одиницями	17
	2	Знайти в масиві і вивести на екран середнє арифметичне від'ємних елементів кожного рядка масиву	6x7
14	1	Знайти добуток 10-ти перших елементів і суму позосталих елементів масиву	16
	2	Знайти в масиві і вивести на екран суму елементів, розташованих на головній діагоналі	5x5

№ варі-анта	№ завда-ня	Зміст завдання	Розмір-ність
15	1	Знайти суму і кількість елементів, розташованих до першого від’ємного елементу	14
	2	Знайти в масиві і вивести на екран суму абсолютних значень від’ємних елементів масиву	5x7
16	1	Знайти середнє арифметичне додатних елементів і суму від’ємних елементів масиву	18
	2	Знайти в масиві і вивести на екран значення максимального елемента і кількість нульових елементів масив	3x6
17	1	Замінити мінімальний елемент нулем і визначити середнє арифметичне всіх елементів масиву	15
	2	Знайти в масиві і вивести на екран суму елементів, які знаходяться вище головної діагоналі, включаючи елементи діагоналі	6x6
18	1	Знайти суму і добуток від’ємних елементів, а також кількість нульових елементів	17
	2	Знайти в масиві і вивести на екран суму елементів, які лежать вище головної діагоналі, без елементів головної діагоналі	7x7
19	1	Знайти добуток додатних елементів з парними індексами і суму елементів з непарними індексами	17
	2	Знайти в масиві і вивести на екран суму елементів, які лежать нижче головної діагоналі, включаючи елементи головної діагоналі	6x6
20	1	Знайти суму всіх елементів, попередньо замінивши всі від’ємні елементи нулями	13
	2	Знайти в масиві і вивести на екран суму елементів, які лежать нижче головної діагоналі, не включаючи елементи головної діагоналі	5x5
21	1	Знайти суму квадратів від'ємних елементів і їх кількість	15
	2	Знайти в масиві і вивести на екран добуток всіх додатних елементів (крім нульових)	3x7
22	1	Знайти середнє арифметичне натуральних логарифмів додатних елементів і їх кількість	13
	2	Знайти в масиві і вивести на екран максимальний і мінімальний елементи, номери їх рядків і стовпців, а також поміняти їх місцями	4x6
23	1	Знайти максимальне значення серед елементів з парними індексами і мінімальне значення серед елементів з непарними індексами	18
	2	Знайти в масиві і вивести на екран найбільші значення елементи кожного рядка і записати їх в масив X	4x6

№ варі-анта	№ завда-ня	Зміст завдання	Розмір-ність
24	1	Знайти середнє геометричне додатних елементів і їх кількість (середнє геометричне визначається за формулою $p = \sqrt[k]{\prod_{i=1}^{k} x_i}$, де k – кількість додатних елементів)	19
	2	Знайти в масиві і вивести на екран мінімальний елемент і записати нулі в той рядок і той стовпець, до яких він відноситься	7x4
25	1	Знайти середнє гармонійне елементів, менших за нуль ($x_i < 0$) та їх кількість. Середнє гармонійне обчислюється за формулою: $$G = \frac{1}{\frac{1}{k}\sum_{x_i<0}\frac{1}{x_i}}$$ де k – кількість від'ємних елементів.	15
	2	Знайти в масиві і вивести на екран суму елементів, які лежать нижче головної діагоналі, не включаючи елементи головної діагоналі	5x5

<u>Приклад виконання типового завдання для Лабораторної роботи № 1.</u>

Таблиця 1.2.

Умови завдання

0	1	Знайти два сусідні елементи, сума яких максимальна, і вивести ці елементи в порядку зростання їх індексів.	13
	2	Написати програму для обчислення кількості елементів масиву, відмінних від нуля, а також визначити в масиві P[6,4] добуток додатних елементів кожного рядка.	6x4

Приклад виконання Лабораторної роботи № 1

```
// Умова: Лабораторна робота №1
// Завдання №1 (Приклад виконання)
// Знайти два сусідні елементи, сума яких максимальна,
// і вивести ці елементи в порядку зростання їх індексів

#include <iostream> // підключаємо бібліотеку вводу-виводу
#include <windows.h> // підключаємо заголовний файл локалізації
// підключаємо бібліотеки для отримання випадкових чисел
#include <ctime> // підключаємо бібліотеку з функцією time()
#include <cstdlib> // підключаємо бібліотеку з функціями srand(), rand()

using namespace std; // оголошуємо простір імен std

void printDiap(float [], char [], int, int); // прототип функції виводу

int main() // функція main, яка виконується першою
{
    SetConsoleOutputCP(1251); // локалізуємо вивід

    // описуємо дані:
    const int n = 13; // число елементів масиву
    float arr [n] = {3, 5, 7, 1, 10, 23,      // ініціалізуємо масив
                     3, 8, 9, 34, 5, 16, 22}; //
    char name[] = "arr";  // ім'я масиву для виводу

    // перевіряємо вміст масиву arr
    cout << "Вміст масиву arr:" << endl; // виводимо заголовок
    printDiap(arr, name, 0, n-1);        // виводимо масив
    cout << endl;                        // переводимо рядок

    // заповнюємо масив arr випадковими числами
    srand(time(0)); // встановлюємо автоматичну рандомізацію
    for (int i=0; i<n; i++)       // вводимо випадкові цілі числа
        arr[i] = 1 + rand()% 100; // довжиною не більше двох цифр

    // перевіряємо вміст масиву arr
    cout << "Випадкові числа в масиві arr:" << endl; // виводимо заголовок
    printDiap(arr, name, 0, n-1);   // виводимо масив
    cout << endl;                   // переводимо рядок

    // вводимо в масив arr числа з клавіатури
    cout << "Введіть в масив arr числа з клавіатури:" << endl;
    for (int i=0; i<n; i++)
    {
        cout << "arr[" << i << "] = "; // виводимо запрошення
        cin >> arr[i];                 // вводимо масив
    }
    cout << endl; // переводимо рядок
```

```
    // перевіряємо вміст масиву arr
    cout << "Вміст масиву arr:" << endl; // виводимо заголовок
    printDiap(arr, name, 0, n-1); // виводимо масив
    cout << endl << endl;         // переводимо рядок

    // вирішуємо завдання
    // опис даних:
    float sum;    // сума сусідніх елементів
    int indFirst, // індекс першого елемента
    indSecond;    // індекс сусіднього елемента

    // вважаємо найбільшою суму 1-го і 2-го ел-тів
    sum = arr[0] + arr[1];
    indFirst = 0;  // тоді,
    indSecond = 1; // відповідно

    // тепер перевіряємо інші поєднання
    for (int i=1; i<n-1; i++) // починаємо з 2-го, якщо їх сума
         if ( arr[i] + arr[i + 1] > sum ) // перевищує попередню
         {
             sum = arr[i] + arr[i + 1]; // оновлюємо суму
             indFirst = i;              // оновлюємо їх
             indSecond = i + 1;         // індекси
         }
    // виводимо результат
    cout << "Сума цих елементів є максимальною:" << endl;
    cout << "arr[" << indFirst << "] = "
         << arr[indFirst] << endl;
    cout << "arr[" << indSecond << "] = "
         << arr[indSecond] << endl;

    return 0; // успішне закінчення програми
}             // кінець функції main

// функція виводу фрагмента одновимірного масиву
void printDiap ( // опис формальних параметрів:
    float a[],   // масив, що підлягає виводу
    char ch[],   // ім'я масиву
    int ii,      // початковий індекс виведення
    int in       // кінцевий індекс виведення
){
    int forCout = 0; // інтервал переводу рядка

    // цикл виведення елементів масиву
    for (int j = ii; j <= in; j++)
    {
        cout << ch << '[' << j << "] = "  // виводимо
             << a[j] << "  ";             // в рядок
        if ( forCout == 4 )  // якщо інтервал досягнуто
        {
            cout << endl;    // переводимо рядок
```

```
            forCout = 0;     // обнуляємо змінну інтервала
        }
        else                 // інакше
            ++forCout;       // збільшуємо змінну інтервала
    }
    cout << endl;            // переводимо рядок
}
```

```
Вміст масиву arr:
arr[0] = 3   arr[1] = 5   arr[2] = 7   arr[3] = 1   arr[4] = 10
arr[5] = 23   arr[6] = 3   arr[7] = 8   arr[8] = 9   arr[9] = 34
arr[10] = 5   arr[11] = 16   arr[12] = 22

Випадкові числа в масиві arr::
arr[0] = 93   arr[1] = 60   arr[2] = 17   arr[3] = 42   arr[4] = 41
arr[5] = 28   arr[6] = 98   arr[7] = 60   arr[8] = 85   arr[9] = 46
arr[10] = 87   arr[11] = 89   arr[12] = 96

Введіть в масив arr числа з клавіатури:
```

Рис. 1.2. Результат виконання програми

```
arr[0] = 6
arr[1] = 23
arr[2] = 44
arr[3] = 56
arr[4] = 27
arr[5] = 51
arr[6] = 8
arr[7] = 44
arr[8] = 63
arr[9] = 72
arr[10] = 33
arr[11] = 45
arr[12] = 65

Вміст масиву arr:
arr[0] = 6   arr[1] = 23   arr[2] = 44   arr[3] = 56   arr[4] = 27
arr[5] = 51   arr[6] = 8   arr[7] = 44   arr[8] = 63   arr[9] = 72
arr[10] = 33   arr[11] = 45   arr[12] = 65

Сума цих елементів є максимальною:
arr[8] = 63
arr[9] = 72
```

Рис. 1.2. Результат виконання програми (продовження)

```
// Умова: Лабораторна робота №1
// Завдання №2 (Приклад виконання)

// Написати програму для обчислення кількості елементів
// масиву, відмінних від нуля, а також визначити в масиві p[6]
// добуток додатних елементів кожного рядка.

#include <iostream> // підключаємо бібліотеку вводу-виводу
#include <windows.h> // підключаємо заголовний файл локалізації
```

```
// підключаємо бібліотеки для отримання випадкових чисел
#include <ctime> // підключаємо бібліотеку з функцією time()
#include <cstdlib> // підключаємо бібліотеку з функціями srand(), rand()

const int n = 6; // кількість рядків масиву
const int m = 4; // кількість стовпців масиву
void printDiap(float [][m], char [], int, int); // прототип функції виводу

using namespace std; // оголошуємо простір імен std

int main()           // функція main виконується першою
{
    SetConsoleOutputCP(1251); // локалізуємо вивід

    // описуємо дані:
    float arr[n][m] = {{3.1, -5.3,5.5,0},    // 1-й рядок масиву
                       {-4.5,2.4, -3.9,5.3}, // 2-й рядок масиву
                       {7.2, -1.6,0,7.6},    // 3-й рядок масиву
                       {6.3,0, -9.7,4.7},    // 4-й рядок масиву
                       {0, -8.6,5.7,3.88},   // 5-й рядок масиву
                       {23.6,7.2,9.7,0}}; // 6-й рядок масиву;
    char name[] = "arr";  // ім'я масиву для виводу

    // перевіряємо вміст масиву arr[n][m]
    cout << "Вміст масиву arr[n][m]:" << endl; // виводимо заголовок
    printDiap(arr, name, n, m);                // виводимо масив

    // вводимо в масив arr[n][m] випадкові дійсні числа
    srand(time(NULL));  // встановлюємо автоматичну рандомізацію
    // заповнюємо матрицю випадковими числами
    for (int i = 0; i <n; i ++) // цикл по рядках
        for (int j = 0; j <m; j ++) // цикл по стовпцях
            arr[i][j] = (float)(rand()% 20001) // діапазон чисел[-100;100]
                          / 100-100;          //з двома знаками після коми

    // перевіряємо вміст масиву arr[n][m]
    cout << "Випадкові дійсні числа в масиві arr[n][m]:" << endl;
    printDiap(arr, name, n, m);  // виводимо масив

    // вводимо в масив arr[n][m] числа з клавіатури
    cout << "Введіть в масив arr[n][m] числа з клавіатури:" << endl;
    for (int i = 0; i <n; i ++)           // цикл по рядках
    for (int j = 0; j <m; j ++)           // цикл по стовпцях
    {
        cout << "arr[" << i << "][" << j  // запрошуємо
             << "] = ";                    // до введення
        cin >> arr[i][j];                 // вводимо
    }
    cout << endl;  // переводимо рядок

    // перевіряємо вміст масиву arr[n][m]
```

```
    cout << "Дійсні числа в масиві arr[n][m]:" << endl;
    printDiap(arr, name, n, m);   // виводимо масив

    // виконуємо завдання:
    // опис даних:
    float p[n];          // масив добутків додатних
                         // елементів кожного рядка матриці
    int numNonZero = 0;  // кількість елементів матриці
                         // відмінних від 0

    // всі обчислення здійснюємо одночасно
    for (int i = 0; i <n; i ++) // цикл по рядках
    {
        p[i] = 1.0; // перед множенням заносимо 1.0
        for (int j = 0; j <m; j ++) // цикл по стовпцях
        {
            if ( arr[i][j] != 0 )  // якщо елемент ненульовий
                ++ numNonZero;     // збільшуємо кількість на 1
            if ( arr[i][j] > 0 )   // якщо елемент додатний
                p[i] *= arr[i][j]; // множимо на нього
        }
    }

    // виводимо результати
    cout << "Кількість елементів, відмінних від нуля = "
         << numNonZero << endl;

    cout << "Добуток додатних елементів:" << endl;
    for (int i = 0; i <n; i ++)  // цикл по масиву p[i]
        cout << "в рядку " << i << " = " << p[i] << endl;
    cout << endl; // переводимо рядок

    return 0; // успішне закінчення програми
}             // кінець функції main


// функція виводу двовимірного масиву
void printDiap (  // опис формальних параметрів:
    float a[][m], // масив, що підлягає виводу
    char ch[],    // ім'я масиву
    int n,        // кількість рядків масиву
    int m         // кількість стовпців масиву
){
    int forCout = 0; // інтервал переводу рядка

    // цикл виведення елементів масиву
    for (int i=0; i<n; i++)
        for (int j=0; j<m; j++)
        {
            cout << ch << '[' << i << "]["  // виводимо
                 << j << "] = " << a[i][j]  // масив
```

```
                << "   ";                    // в рядок
            if ( forCout == 3 )  // якщо інтервал досягнуто
            {
                cout << endl;    // переводимо рядок
                forCout = 0;     // обнуляємо змінну інтервала
            }
            else                 // інакше
                ++forCout;       // збільшуємо змінну інтервал
    }
    cout << endl;            // переводимо рядок
}
```

```
Вміст масиву arr[n][m]:
arr[0][0] = 3.1  arr[0][1] = -5.3  arr[0][2] = 5.5  arr[0][3] = 0
arr[1][0] = -4.5  arr[1][1] = 2.4  arr[1][2] = -3.9  arr[1][3] = 5.3
arr[2][0] = 7.2  arr[2][1] = -1.6  arr[2][2] = 0  arr[2][3] = 7.6
arr[3][0] = 6.3  arr[3][1] = 0  arr[3][2] = -9.7  arr[3][3] = 4.7
arr[4][0] = 0  arr[4][1] = -8.6  arr[4][2] = 5.7  arr[4][3] = 3.88
arr[5][0] = 23.6  arr[5][1] = 7.2  arr[5][2] = 9.7  arr[5][3] = 0

Випадкові дійсні числа в масиві arr[n][m]:
arr[0][0] = -87.57  arr[0][1] = -24.78  arr[0][2] = 78.7  arr[0][3] = -74.5
arr[1][0] = -33.6  arr[1][1] = -44.39  arr[1][2] = -28.12  arr[1][3] = -10.85
arr[2][0] = -22.71  arr[2][1] = 35.82  arr[2][2] = -13.62  arr[2][3] = 28.81
arr[3][0] = 54.85  arr[3][1] = -26.37  arr[3][2] = -1.48  arr[3][3] = -39.95
arr[4][0] = 67.5  arr[4][1] = -79.71  arr[4][2] = 69.95  arr[4][3] = -78.04
arr[5][0] = 79.17  arr[5][1] = -72.24  arr[5][2] = 22.22  arr[5][3] = 30.43
```

Рис. 1.3. Результат виконання програми

```
Введіть в масив arr[n][m] числа з клавіатури:
arr[0][0] = 2.5
arr[0][1] = -46.7
arr[0][2] = 0
arr[0][3] = 34.6
arr[1][0] = 2.7
arr[1][1] = -72.6
arr[1][2] = 36.7
arr[1][3] = 0
arr[2][0] = -54.12
arr[2][1] = 43.6
arr[2][2] = 4.8
arr[2][3] = 73.66
arr[3][0] = -82.33
arr[3][1] = 0
arr[3][2] = 12.92
arr[3][3] = 16.63
arr[4][0] = 5.88
arr[4][1] = -33.22
arr[4][2] = -18.4
arr[4][3] = 54.7
arr[5][0] = 11.56
arr[5][1] = 8.07
arr[5][2] = 0
arr[5][3] = -92.7
```

Рис. 1.3. Результат виконання програми (продовження)

```
Дійсні числа в масиві arr[n][m]:
arr[0][0] = 2.5  arr[0][1] = -46.7  arr[0][2] = 0  arr[0][3] = 34.6
arr[1][0] = 2.7  arr[1][1] = -72.6  arr[1][2] = 36.7  arr[1][3] = 0
arr[2][0] = -54.12  arr[2][1] = 43.6  arr[2][2] = 4.8  arr[2][3] = 73.66
arr[3][0] = -82.33  arr[3][1] = 0  arr[3][2] = 12.92  arr[3][3] = 16.63
arr[4][0] = 5.88  arr[4][1] = -33.22  arr[4][2] = -18.4  arr[4][3] = 54.7
arr[5][0] = 11.56  arr[5][1] = 8.07  arr[5][2] = 0  arr[5][3] = -92.7

Кількість елементів, відмінних від нуля = 20
Добуток додатних елементів:
в рядку 0 = 86.5
в рядку 1 = 99.09
в рядку 2 = 15415.6
в рядку 3 = 214.86
в рядку 4 = 321.636
в рядку 5 = 93.2892
```

Рис. 1.3. Результат виконання програми (продовження)

ЛАБОРАТОРНА РОБОТА № 2

Пошук мінімальних і максимальних значень елементів (ключів) в одновимірних і двовимірних динамічних масивах.

Мета роботи: опанування принципів роботи з динамічними масивами, заповненими значеннями даних різних типів.

Теоретичні відомості. Динамічні одновимірні масиви в С ++

Для створення динамічних масивів в С ++ використовуються покажчики. Покажчик – це змінна, значенням якої є адреса комірки пам'яті, що зберігає дані певного типу. Зазвичай покажчик посилається на блок даних з області пам'яті, причому саме на його початок.

Покажчики різняться за типом в залежності від об'єктів, на які вони вказують. Під розміщення покажчиків всіх типів виділяється однаковий обсяг пам'яті, тому що розмір адреси залежить тільки від обчислювальної системи. Це зроблено для уникнення спроб присвоєння даних, що не відповідають типу покажчика, наприклад, змінної типу `double` за адресою покажчика `int` (тип `double` складається з 8-ми байт, а `int` – з 4-х!).

Для роботи з адресами об'єктів в С ++ використовуються дві операції:

❶. `&` – операція взяття адреси об'єкта.
❷. `*` – операція звернення за адресою об'єкта, на який вказує покажчик.

наприклад:

```
int *yPtr;    // оголошуємо покажчик на змінну цілого типу -
              // тут * вказує на тип змінної, yPtr – ім'я (адреса) змінної
int y = 5;    // оголошуємо статичну змінну цілого типу зі значенням 5, тут
              // y – ім'я (адреса) змінної (ім'я для нас, а адреса – для компілятора!)
yPtr = &y;                                        // заносимо адресу y в покажчик
cout << "Адреса y =" << &y << endl;               // виводимо адресу y
cout << "Адреса y в yPtr =" << yPtr << endl;      // виводимо адресу y
cout << "Вміст y =" << *yPtr << endl;             // виводимо вміст y = 5
*yPtr = 10;                                       // тепер до y заносимо 10
cout << "Вміст y =" << y << endl;                 // виводимо вміст y = 10
delete yPtr;                                      // звільняємо пам'ять
```

Динамічна пам'ять (ДП) виділяється операційною системою (ОС) за запитом програміста за допомогою спеціальних інструкцій. Це дозволяє в процесі роботи програми коригувати обсяг використовуваної пам'яті.

Динамічна пам'ять в мові С++ під відповідні типи даних (`float, int, char` та ін.) з використанням покажчиків виділяється за допомогою оператора `new`, А звільняється – за допомогою оператора `delete`.

наприклад:

```
int *d = new int;                // оголошуємо покажчик d на ціле число і виділяємо
                                 // в пам'яті динамічну змінну, на яку він вказує
*D = 10;                         // заносимо в динамічну змінну  d  число  10
cout << "*d =" << *d << endl;    // виведення вмісту динамічної змінної
deleted;                         // обов'язково звільняємо пам'ять
```

Виділення динамічної пам'яті для одновимірного масиву здійснюється таким чином:

```
int n;  // розмір одновимірного динамічного масиву
cout << "Введіть розмір масиву n =";
cin >> n;  // вводимо значення  n
// створюємо динамічний одновимірний масив
int *pArr = new int[n];  // отримуємо покажчик на динамічний масив
                         // і сам масив з  n  цілих елементів!!!
// заповнюємо динамічний одновимірний масив
for (int i = 0; i < n; i ++)  // використовуємо значення  i
    pArr[i] = i;  // тут адреса кожного елемента дорівнює вихідному
                  // значенню покажчика + значення індексу
// після завершення всіх дій з масивом
delete[] pArr;  // обов'язково звільняємо динамічну пам'ять
```

Двовимірні динамічні масиви в С ++

У разі створення двовимірного динамічного масиву виділення пам'яті здійснюється наступним способом (див. рис. 2.1):

```
int nArr,  // кількість рядків динамічного двовимірного масиву
    mArr;  //кількість стовпців динамічного двовимірного масиву
 cout << "Для динамічного двовимірного масиву" << endl
 << "введіть кількість рядків =";
 cin >> N;  // вводимо кількість рядків
 cout << "і стовпців =";
 cin >> M;   // вводимо кількість стовпців
 float **a = new float *[nArr];   // оголошуємо змінну типу  «покажчик
                                  // на покажчик на  float»  і виділяємо пам'ять
                                  // під цей масив покажчиків
// тепер «вкажемо» кожним з покажчиків отриманого масиву на масив
// типу  float  розмірності  mArr:
for (int i = 0; i < nArr; i++)  // організуємо цикл для виділення
                                // рядків масиву за кількістю стовпців,
    a[i] = new float[mArr];     // де кожен елемент масиву покажчиків
                                // вказує на масив рядків, причому
                                // адреса кожного елемента –  a[i][j]
// ================================================= =================
```

Видаляється такий масив у зворотному порядку:

```
for (int i=0; i < nArr; i++)
    delete [] a[i];  // спочатку видаляємо масиви значень
delete [] a;         // потім видаляємо масив покажчиків
```

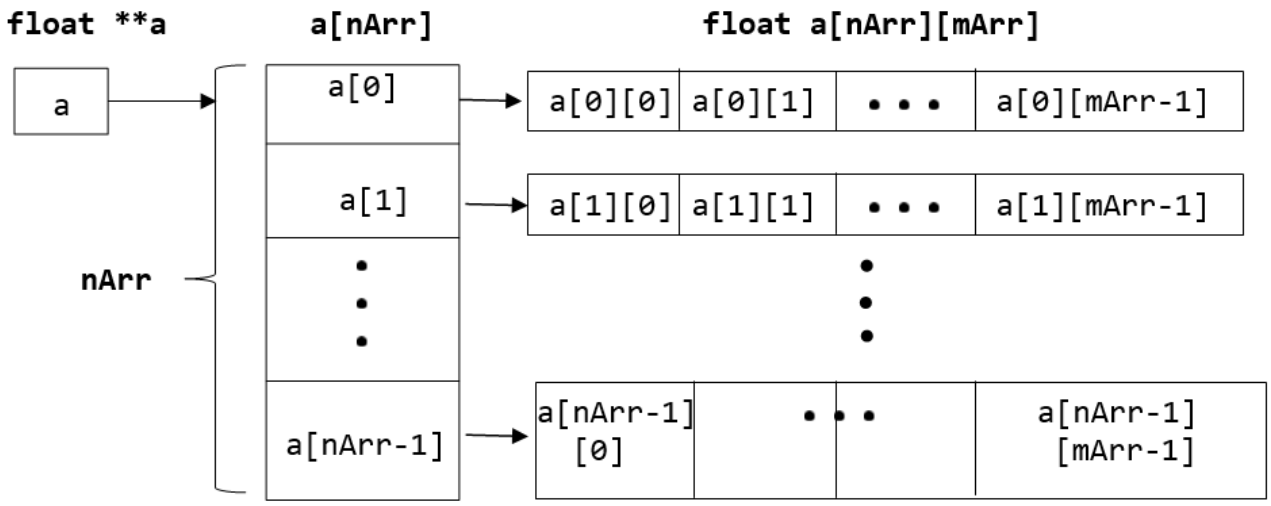

Рис. 2.1. Розташування компонентів в процесі створення двовимірного динамічного масиву

<u>Завдання до Лабораторної роботи № 2.</u>

Розробити програму для опису одновимірних і двовимірних динамічних масивів відповідно до заданого у таблиці 2.1 варіанту і виконати необхідні розрахунки.

У розробленій програмі виконати наступні дії.

1. Вибрати тип масиву в залежності від номера отриманого завдання і ввести його розмірність з клавіатури!
2. Створити **динамічний** масив відповідної розмірності.
3. Ввести в масив дані з клавіатури.
4. Перевірити вміст масиву в завданні №1 (за допомогою циклу `for` і операції виводу `cout`).
5. Виконати зазначені в завданні №1 дії і вивести результати. Зробити висновки.

Повторити пункти 1-5 для масиву №2. Зробити висновки.

Текст програми необхідно оформити та супроводити коментарями відповідно до Вимог.

Занести у звіт отримані результати.

Таблиця 2.1

Варіанти завдань до Лабораторної роботи № 2

№ варі-анту	№ завдан-ня	Зміст завдання	Розмір-ність
1	1	Дан цілочисельний масив A розміру N, всі елементи якого, крім першого, впорядковані за зростанням. Зробити масив упорядкованим, перемістивши перший елемент на нову позицію A_X. Всі елементи правіше A_X розмістити в новому динамічному масиві. Вивести розмір і вміст масиву.	9

№ варі-анту	№ завдан-ня	Зміст завдання	Розмір-ність
	2	Дан масив дійсних чисел A розміру $N \times M$. Значення всіх його елементів без максимального елемента скопіювати в одновимірний динамічний масив.	4x5
2	1	Дан цілочисельний масив розміру N. Переставити в зворотному порядку елементи масиву, розташовані між його мінімальним і максимальним елементами, включаючи мінімальний і максимальний елементи і розмістити в новому динамічному масиві. Вивести розмір і вміст масиву.	7
	2	Дан масив дійсних чисел A розміру $N \times N$. Сформувати одновимірний динамічний масив з елементів вихідного масиву, розташованих над головною діагоналлю.	5x5
3	1	Дан цілочисельний масив розміру N. Елементи масиву, розташовані між його мінімальним і максимальним елементами (не включаючи мінімальний і максимальний елементи) розмістити в новому динамічному масиві. Вивести розмір і вміст масиву.	7
	2	Дан масив дійсних чисел A розміру $N \times M$. Всі його елементи, ціла частина яких кратна трьом, записати в динамічний одновимірний масив.	6x4
4	1	Дан цілочисельний масив A розміру N і цілі числа K і L ($1 \le K < L \le N$). Переставити в зворотному порядку елементи масиву, розташовані між елементами A_K і A_L, включаючи ці елементи, і розмістити в новому динамічному масиві. Вивести розмір і вміст масиву.	11
	2	Дан масив дійсних чисел A розміру $N \times N$. Сформувати одновимірний динамічний масив з елементів вихідного масиву, розташованих під головною діагоналлю.	4x4
5	1	Дан цілочисельний масив розміру N. Збільшити всі парні числа, що містяться в масиві, на початкове значення першого парного числа і розмістити в новому динамічному масиві. Вивести розмір і вміст масиву. Якщо парні числа в масиві відсутні, то залишити масив без змін і видати відповідне повідомлення.	8
	2	Дан масив дійсних чисел A розміру $N \times M$. Значення всіх його додатних елементів записати в одновимірний динамічний масив.	5x4

№ варі-анту	№ завдан-ня	Зміст завдання	Розмір-ність
6	1	Дан цілочисельний масив *A* розміру *N*. Сформувати новий динамічний масив *B*, в який записати всі додатні елементи масиву *A*, зберігаючи вихідний порядок проходження елементів. Вивести розмір і вміст масиву *B*.	7
	2	Дан масив дійсних чисел *A* розміру *N*×*M*. Знайти в ньому максимальний елемент. Наступні за ним елементи записати в одновимірний динамічний масив.	6x4
7	1	Дан цілочисельний масив розміру *N*. Переписати в новий цілочисельний динамічно заданий масив *B* всі парні значення елементів з вихідного масиву (в тому ж порядку) і вивести розмір отриманого масиву *B* і його вміст.	6
	2	Дан цілий масив *A* розміру *N*×*M*. Значення всіх його парних елементів записати в одновимірний динамічний масив.	4x5
8	1	Дан цілочисельний масив розміру *N*. Якщо він є *перестановкою*, тобто містить всі числа від 1 до *N*, то вивести 0, а в іншому випадку вивести номер першого неприпустимого елемента. Всі елементи після першого неприпустимого елемента розмістити в пам'яті динамічно і роздрукувати.	13
	2	Дан масив дійсних чисел *A* розміру *N*×*M*. Всі його від’ємні елементи записати в динамічний одновимірний масив.	5x6
9	1	Дан цілочисельний масив розміру *N*. Якщо він є *перестановкою*, тобто містить всі числа від 1 до *N*, то вивести 0, а в іншому випадку вивести номер першого неприпустимого елемента. Всі елементи до першого неприпустимого елемента розмістити в пам'яті динамічно і роздрукувати.	12
	2	Дан масив дійсних чисел *A* розміру *N*×*M*. Значення всіх його елементів без мінімального елемента скопіювати в одновимірний динамічний масив.	5x5
10	1	Дано число *R* і цілочисельний масив розміру *N*. Знайти два різних елемента масиву, сума яких найбільш близька до числа *R*, і вивести ці елементи, а елементи, які знаходяться між ними, розмістити в пам'яті динамічно і роздрукувати. Якщо елементи знаходяться поруч – видати відповідне повідомлення.	7

№ варі-анту	№ завдан-ня	Зміст завдання	Розмір-ність
	2	Дан масив дійсних чисел *A* розміру *N×M*. Всі його елементи, ціла частина яких кратна чотирьом, записати в динамічний одновимірний масив.	4x6
11	1	Дан цілочисельний масив розміру *N*. Знайти номери двох найближчих елементів з цього масиву (тобто елементів з найменшим модулем різниці) і вивести ці номери. Елементи між двома найближчими елементами розмістити в пам'яті динамічно і роздрукувати. Якщо елементи знаходяться поруч – видати відповідне повідомлення.	14
	2	Дан масив дійсних чисел *A* розміру *N×N*. Всі його елементи, розташовані на головній діагоналі і нижче неї, записати в динамічний одновимірний масив.	6x6
12	1	Дан цілочисельний масив розміру *N* і цілі числа *K* і *L* ($1 \le K \le L \le N$). Знайти суму елементів масиву з номерами від *K* до *L* включно, а самі елементи динамічно розмістити в пам'яті і роздрукувати.	9
	2	Дан масив дійсних чисел *A* розміру *N×M*. Знайти в ньому всі елементи, які знаходяться за межами інтервалу $[c_1, c_2]$, і занести їх до динамічного масиву. Значення c_1 і c_2 задаються самостійно.	6x4
13	1	Дан масив розміру *N* і цілі числа *K* і *L* ($1 \le K \le L \le N$). Знайти середнє арифметичне елементів масиву з номерами від *K* до *L* включно, а елементи, що не входять в цей діапазон, динамічно розмістити в пам'яті і роздрукувати.	12
	2	Дан масив дійсних чисел *A* розміру *N×M*. Знайти в ньому мінімальний елемент. Наступні за ним елементи записати в одновимірний динамічний масив.	5x4
14	1	Дан цілочисельний масив розміру *N* і цілі числа *K* і *L* ($1 < K \le L \le N$). Знайти суму всіх елементів масиву, крім елементів з номерами від *K* до *L* включно. Елементи з номерами від 1 до *K* динамічно розмістити в пам'яті і роздрукувати.	10
	2	Дан масив дійсних чисел *A* розміру *N×M*. Значення всіх його від'ємних елементів записати в одновимірний динамічний масив.	7x3
15	1	Дан цілочисельний масив розміру *N*, в якому чередуються парні і непарні числа. Визначити кількість непарних чисел і розмістити їх динамічно в пам'яті і роздрукувати.	15

№ варі-анту	№ завдан-ня	Зміст завдання	Розмір-ність
	2	Дан масив дійсних чисел *A* розміру *N×N*. Всі його елементи, розташовані на побічній діагоналі і нижче неї, записати в динамічний одновимірний масив.	6x6
16	1	Дан цілочисельний масив *A* розміру *N*. Знайти мінімальний елемент з його елементів з парними номерами: *A[2]*, *A[4]*, *A[6]*, ... Всі елементи масиву до мінімального елемента розмістити динамічно в пам'яті і роздрукувати.	11
	2	Дан масив дійсних чисел *A* розміру *N×N*. Всі його елементи, розташовані на побічній діагоналі і вище неї, записати в динамічний одновимірний масив.	5x5
17	1	Дан цілочисельний масив *A* розміру *N*. Знайти мінімальний елемент з його елементів з парними номерами: *A[2]*, *A[4]*, *A[6]*, ... Всі елементи масиву після мінімального елемента розмістити динамічно в пам'яті і роздрукувати.	17
	2	Дан масив дійсних чисел *A* розміру *N×M*. Значення його елементів, що задовольняють умові $p_1<=A[i][j]<=p_2$ розмістити в динамічної пам'яті. Значення p_1 і p_2 задаються самостійно.	6x5
18	1	Дан цілочисельний масив розміру *N*. Знайти номери тих елементів масиву, які більші від свого правого сусіда, і кількість таких елементів. Знайдені номери розмістити динамічно в пам'яті і роздрукувати.	14
	2	Дан масив дійсних чисел *A* розміру *N×N*. Всі його елементи, розташовані на головній діагоналі і вище неї, записати в динамічний одновимірний масив.	6x6
19	1	Дан цілочисельний масив розміру *N*. Знайти номери тих елементів масиву, які більше від свого лівого сусіда, і кількість таких елементів. Знайдені номера розмістити динамічно в пам'яті і роздрукувати.	11
	2	Дан цілий масив *A* розміру *N×M*. Значення всіх його непарних елементів записати в одновимірний динамічний масив.	4x7
20	1	Дан масив розміру *N*. Знайти номер його першого локального мінімуму (локальний мінімум – це елемент, який менший від будь-якого зі своїх сусідів). Елементи, що знаходяться правіше локального мінімуму, розмістити в пам'яті динамічно і роздрукувати.	16

№ варі-анту	№ завдан-ня	Зміст завдання	Розмір-ність
	2	Дан масив дійсних чисел *A* розміру *N*×*M*. Всі його додатні елементи записати в динамічний одновимірний масив.	5x6
21	1	Дан цілочисельний масив розміру *N*. Знайти номер його першого локального мінімуму (локальний мінімум – це елемент, який менший від будь-якого зі своїх сусідів). Елементи, що знаходяться лівіше локального мінімуму, розмістити в пам'яті динамічно і роздрукувати.	10
	2	Дан масив дійсних чисел *A* розміру *N*×*N*. Сформувати одновимірний динамічний масив з додатних елементів вихідного масиву, розташованих під головною діагоналлю.	6x5
22	1	Дан цілочисельний масив розміру *N*. Знайти максимальний з його локальних мінімумів (локальний мінімум – це елемент, який менший від будь-якого зі своїх сусідів). Елементи, що знаходяться правіше максимального локального мінімуму, розмістити в пам'яті динамічно і роздрукувати.	11
	2	Дан масив дійсних чисел *A* розміру *N*×*N*. Сформувати одновимірний динамічний масив з елементів вихідного масиву, розташованих над побічною діагоналлю.	6x6
23	1	Дан цілочисельний масив розміру *N*. Знайти максимальний з його локальних мінімумів (локальний мінімум – це елемент, який менший від будь-якого зі своїх сусідів). Елементи, що знаходяться лівіше максимального локального мінімуму, розмістити в пам'яті динамічно і роздрукувати.	13
	2	Дан масив дійсних чисел *A* розміру *N*×*M*. Всі його елементи, ціла частина яких кратна п'яти, записати в динамічний одновимірний масив.	5x6
24	1	Дано число *R* і цілочисельний масив *A* розміру *N*. Знайти елемент масиву, який найбільш близький до числа *R* (тобто такий елемент A_K, для якого величина $\|A_K-R\|$ є мінімальною). Елементи, що знаходяться правіше A_K, розмістити в пам'яті динамічно і роздрукувати.	12

№ варі-анту	№ завдан-ня	Зміст завдання	Розмір-ність
	2	Дан масив дійсних чисел A розміру $N \times N$. Сформувати одновимірний динамічний масив з додатних елементів вихідного масиву, розташованих над головною діагоналлю.	6x5
25	1	Дано число R і цілочисельний масив A розміру N. Знайти елемент масиву, який найбільш близький до числа R (тобто такий елемент A_K, для якого величина $\|A_K-R\|$ є мінімальною). Елементи, що знаходяться лівіше A_K, розмістити в пам'яті динамічно і роздрукувати.	19
	2	Дан масив дійсних чисел A розміру $N \times N$. Сформувати одновимірний динамічний масив з елементів вихідного масиву, розташованих під побічною діагоналлю.	5x5

<u>Приклад виконання типового Завдання Лабораторної роботи №2.</u>

Таблиця 2.2.

Умови завдання

0	1	Дан цілочисельний масив A розміру N. Знайти в цьому масиві максимальний і мінімальний елементи і всі елементи, що знаходяться між ними помістити в динамічний масив.	9
	2	Дан масив дійсних чисел A розміру $N \times M$. Сформувати одновимірний динамічний масив з елементів вихідного масиву, розташованих до максимального елемента.	6x4

```
// Лабораторна робота №2
//
// Виконання завдання №1
//
// Дан цілочисельний масив A розміру N. Знайти в цьому масиві
// максимальний і мінімальний елементи і всі елементи,
// які знаходяться між ними і помістити в динамічний масив

#include <iostream> // підключаємо бібліотеку вводу-виводу
#include <cmath> // підключаємо бібліотеку мат. функцій
#include <windows.h> // підключаємо заголовний файл локалізації

using namespace std; // оголошуємо простір імен std
```

```
int main ()  // функція main виконується першою
{
    SetConsoleOutputCP(1251); // локалізуємо вивід

    // опис даних:
    int N;  // розмір одновимірного динамічного масиву
    cout << "Введіть розмір масиву N = ";
    cin >> N;  // вводимо значення N

    // створюємо динамічний одновимірний масив
    int *A = new int [N]; // отримуємо покажчик на динамічний масив
                          // і сам масив з N цілих елементів !!!
    //
    // вводимо в масив A числа з клавіатури
    cout << "Введіть в масив A цілі числа з клавіатури:" << endl;
    for (int i=0; i<N; i++) // тут адреса кожного елемента дорівнює
    {                             // початковому значенню покажчика
                                  // + значення індексу
        cout << "A[" << i << "] = "; // запрошення
        cin >> A[i];                // введення
    }
    cout << endl; // переводимо рядок

    // перевіряємо вміст масиву A
    cout << "Вміст масиву A:" << endl;
    for (int i=0; i<N; i++)
        cout << "A[" << i << "] = " // виводимо з індексом
             << A[i] << "  ";       // і доповнюємо двома пробілами
    cout << endl << endl; // двічі переводимо рядок

    // виконуємо завдання:
    // опис даних:
    int max,  // значення максимального елемента
        iMax, // індекс максимального елементу
        min,  // значення мінімального елемента
        iMin, // індекс мінімального елемента
        n;    // розмірність динамічного масиву
              // для елементів між max і min

    max = A[0]; // вважаємо, що перший елемент масиву A = max
    iMax = 0;   // відповідно, його індекс дорівнює 0
    min = A[0]; // вважаємо, що перший елемент масиву A = min
    iMin = 0;   // відповідно, його індекс дорівнює 0

    // шукаємо одночасно max і min в масиві A
    for (int i=1; i<N; i++) // починаємо з 2-го елемента
    {
        if ( A[i] > max ) // якщо елемент більше max
        {
            max = A[i];  // у нас новий max
            iMax = i;    // запам'ятовуємо його індекс
```

```
    }
    if ( A[i] < min) // якщо елемент менше min
    {
        min = A[i]; // у нас новий min
        iMin = i;   // запам'ятовуємо його індекс
    }
}
cout << "max = " << max << endl;   // виводимо max
cout << "iMax = " << iMax << endl; // виводимо iMax
cout << "min = " << min << endl;   // виводимо min
cout << "iMin = " << iMin << endl;  // виводимо iMin
cout << endl;                      // переводимо рядок

// вводимо нові змінні
int iStart, // індекс першого елемента в A для перенесення в a
    iEnd,   // індекс останнього елемента в A для перенесення в a
    k = 0;  // індекс в новому динамічному масиві a

// перевіряємо: max і min збігаються, поруч, iMin < iMax
if ( abs(iMax-iMin) > 1 )
{
    // шукаємо, який індекс менший
    if ( iMax > iMin ) // якщо iMin менший
    {
        iStart = iMin + 1; // індекс для початку
        iEnd = iMax;       // індекс для кінця
    }

    else // якщо iMax менший
    {
        iStart = iMax + 1; // індекс для початку
        iEnd = iMin;       // індекс для кінця
    }

    n = abs(iMax-iMin)-1; // обчислюємо розмір динамічного масиву
}
else       // набір даних невдалий
    n = 0; // динамічний масив відводиться не будемо

// якщо набір даних вдалий
if ( n != 0 )
{
    int *a = new int [n]; // отримуємо покажчик на динамічний масив
                          // і сам масив з n цілих елементів !!!
    cout << "Елементи динамічного масиву:" << endl;
    for (int i=iStart; i<iEnd; i++) // цикл по дин. масиву
    {
        a[k] = A[i];               // переносимо елементи з A
        cout << "a[" << k << "] = " // одночасно
             << a[k] << "  ";        // роздруковуємо їх
        ++k;  // просуваємо індекс в дин. масиві
```

```
        }
        delete [] a; // обов'язково звільняємо динамічну пам'ять
    }
    else
        cout << "Масив не може бути отриманий!" << endl;

    cout << endl; // переводимо рядок

    return 0; // успішне закінчення програми
}             // кінець функції main
```

```
Введіть розмір масиву n = 9
Введіть в масив A цілі числа з клавіатури:
A[0] = 2
A[1] = 3
A[2] = 1
A[3] = 4
A[4] = 5
A[5] = 6
A[6] = 7
A[7] = 8
A[8] = 9

Вміст масиву A:
A[0] = 2  A[1] = 3  A[2] = 1  A[3] = 4  A[4] = 5  A[5] = 6  A[6] = 7  A[7] = 8
A[8] = 9

max = 9
iMax = 8
min = 1
iMin = 2

Елементи динамічного масиву:
a[0] = 4  a[1] = 5  a[2] = 6  a[3] = 7  a[4] = 8
```

Рис. 2.2. Результат виконання програми

```
// Лабораторна робота №2
//
// Виконання завдання №2
//
// Дан масив дійсних чисел A розміру NxM. сформувати
// одновимірний динамічний масив з елементів вихідного
// масиву, розташованих до максимального елемента.

#include <iostream> // підключаємо бібліотеку вводу-виводу
#include <windows.h> // підключаємо заголовний файл локалізації

using namespace std; // оголошуємо простір імен std

int main() // функція main виконується першою
{
```

```
SetConsoleOutputCP(1251); // локалізуємо вивід

// опис даних:
int N, // кількість рядків динамічного двовимірного масиву
    M; // кількість стовпців динамічного двовимірного масиву
cout << "Для динамічного двовимірного масиву" << endl
     << "введіть кількість рядків = ";
cin >> N;  // вводимо кількість рядків
cout << "               і стовпців = ";
cin >> M;  // вводимо кількість стовпців
cout << endl; // переводимо рядок

float **A = new float *[N];  // оголошуємо змінну типу «покажчик
                             // на покажчик на float» і виділяємо
                             // пам'ять під цей масив покажчиків

// тепер «вкажемо» кожним з покажчиків отриманого масиву
// на масив типу float розмірності M:
for (int i=0; i<N; i++) // організуємо цикл для виділення
                             // рядків масиву по числу стовпців
    A[i] = new float [M]; // кожен елемент масиву покажчиків
                          // вказує на масив рядків, причому
                          // адреса кожного елемента – A[i][j]

// вводимо в масив A[N][M] числа з клавіатури
cout << "Введіть в масив A[N][M] дійсні числа з клавіатури:" << endl;
for (int i=0; i<N; i++)          // цикл по рядках
    for (int j = 0; j < M; j ++) // цикл по стовпцях
    {
        cout << "A[" << i << "][" << j // запрошуємо
             << "] = ";                // до введення
        cin >> A[i][j];                // вводимо
    }
cout << endl; // переводимо рядок

// перевіряємо вміст масиву A[N][M]
cout << "Дійсні числа в масиві A[N][M]:" << endl;
for (int i=0; i<N; i++) // цикл по рядках
{
    for (int j=0; j<M; j++)            // цикл по стовпцях
        cout << "A[" << i << "]["      // виводимо з індексами
             << j << "] = " << A[i][j] // по рядку і стовпцю
             << "  ";                  // і доповнюємо двома пробілами
    cout << endl; // кожний новий рядок починаємо з нового рядка
}
cout << endl; // переводимо рядок

// виконуємо завдання:
// опис даних – відразу вважаємо, що:
float max = A[0][0];  // значення максимального елемента матриці A
int iA = 0,           // індекс по рядку максимального елемента
```

```
        jA = 0,          // індекс по стовпцю максимального елемента
        qnt = 0,         // число елементів матриці A до max
        k = 0;           // ще нам знадобиться індекс в новому дин. мас.
        bool flag = 0;   // змінна для виходу з подвійного циклу

    // знаходимо max і кількість елементів до нього
    for (int i=0; i<N; i++)      // цикл по рядках
        for (int j=0; j<M; j++)  // цикл по стовпцях
            if ( A[i][j] > max ) // раптом новий елемент A більше max
            {
                max = A[i][j];  // тоді це новий max
                iA = i;         // його індекс по рядку
                jA = j;         // його індекс по стовпцю
                ++qnt;          // і кількість ел-тів до max зросла
            }
    cout << endl; // переводимо рядок

    float *a = new float[qnt]; // отримуємо покажчик на дин. масив
                               // і сам масив з qnt вещ. елементів !!!

    cout << "Елементи одновимірного динамічного масиву:" << endl;
    for (int i=0; i<N; i++) // цикл по рядках дин. масиву A
    {
        for (int j=0; j<M; j++)       // цикл по стовпцях дин. масиву A
            if ( i == iA && j == jA ) // якщо дійшли до max
            {
                flag = 1; // піднімаємо прапор для виходу з циклу по i
                break;    // виходимо з циклу по j
            }
            else
            {
                a[k] = A[i][j]; // переносимо ел-ти матриці в дин. мас.
                cout << "a[" << k << "] = "  // одночасно
                     << a[k] << "  ";        // роздруковуємо їх
                ++k;  // просуваємо індекс в дин. масиві
            }
        if ( flag ) // якщо прапор піднято
            break;  // виходимо з циклу по i
    }

    delete [] a; // обов'язково звільняємо динамічну пам'ять a

    // двовимірний динамічний масив A видаляється в зворотному порядку:
    for (int i=0; i<N; i++)
        delete [] A[i]; // спочатку видаляємо масиви значень
    delete [] A; // потім видаляємо масив покажчиків

    cout << endl; // переводимо рядок

    return 0; // успішне закінчення програми
}             // кінець функції main
```

```
Для динамічного двовимірного масиву
введіть кількість рядків = 6
              і стовпців = 4

Введіть в масив A[N][M] дійсні числа з клавіатури:

A[0][0] = 1
A[0][1] = 2
A[0][2] = 3
A[0][3] = 4
A[1][0] = 5
A[1][1] = 6
A[1][2] = 7
A[1][3] = 8
A[2][0] = 9
A[2][1] = 100
A[2][2] = 1
A[2][3] = 2
A[3][0] = 3
A[3][1] = 4
A[3][2] = 5
A[3][3] = 6
A[4][0] = 7
A[4][1] = 8
A[4][2] = 9
A[4][3] = 1
A[5][0] = 2
A[5][1] = 3
A[5][2] = 4
A[5][3] = 5
Дійсні числа в масиві A[N][M]:
A[0][0] = 1  A[0][1] = 2  A[0][2] = 3  A[0][3] = 4
A[1][0] = 5  A[1][1] = 6  A[1][2] = 7  A[1][3] = 8
A[2][0] = 9  A[2][1] = 100  A[2][2] = 1  A[2][3] = 2
A[3][0] = 3  A[3][1] = 4  A[3][2] = 5  A[3][3] = 6
A[4][0] = 7  A[4][1] = 8  A[4][2] = 9  A[4][3] = 1
A[5][0] = 2  A[5][1] = 3  A[5][2] = 4  A[5][3] = 5

Елементи одновимірного динамічного масиву:
a[0] = 1  a[1] = 2  a[2] = 3  a[3] = 4  a[4] = 5  a[5] = 6  a[6] = 7  a[7] = 8
a[8] = 9
```

Рис. 2.3. Результат виконання програми

ЛАБОРАТОРНА РОБОТА № 3

Формування і реалізація рекурсивних алгоритмів

Мета роботи: ознайомлення з головними особливостями виконання функцій і організації рекурсивних викликів. Розробка алгоритмів рекурсивних обчислень і способи їх реалізації за допомогою мови програмування С++.

Теоретичні відомості. Організація викликів функцій і рекурсії в С ++.

Одним з найважливіших в теорії алгоритмів і в технології програмування є поняття рекурсії і пов'язане з ним мистецтво конструювання працездатних рекурсивних алгоритмів. Розуміння і реалізація питань формування елементів рекурсивних алгоритмів ускладнюються тим, що частина даних алгоритмів реалізується в конструкціях мов програмування і залишається, як би, «за кадром» і явно не проглядається.

Рекурсія *– алгоритмічний прийом, який застосовується для розбиття вихідної задачі на кілька менших підзадач, кожна з яких вирішується за допомогою одного і того ж алгоритму.*

Наприклад, для обходу структури двійкового дерева з однорідними вузлами зручно написати функцію, яка приймає на вхід деякий початковий вузол, виконує для нього необхідні обчислення, а потім викликає саму себе з метою вирішення даного завдання для кожного з пов'язаних з ним його нащадків.

На прикладі знаходження факторіала числа або обчислення різноманітних рядів можна показати, що будь-яка рекурсивна програма легко перетворюється в нерекурсивну, що виконує ті ж обчислення. І навпаки, використовуючи рекурсію, будь-яке обчислення, що припускає застосування циклів, можна реалізувати, не вдаючись до циклів.

Тому при конструюванні алгоритмів слід звертати увагу на три основні проблеми, що виникають при програмуванні рекурсії:

1. *Нескінченна рекурсія.* Будь-рекурсивний алгоритм повинен мати надійні умова зупинки. Можна сформувати рекурсивну функцію, яка ніколи не повертає результат і не досягає кінцевої точки, створюючи нескінченний цикл.

2. *Даремний витрата пам'яті*. При обробці складного ланцюжка рекурсивних звернень програма може вичерпати всю пам'ять стека. Оголошуючи змінні глобально, можна скоротити розмір стека.

3. *Недоречна рекурсія*. Зазвичай це відбувається, коли алгоритм багато разів обчислює одні й ті ж проміжні значення. Для усунення проблем подібного роду можна переписати алгоритм методом «від низу до верху».

Тим не менше, використання рекурсії є красивим й ефективним прийомом програмування, при вивченні якого закріплюються навички формалізованого опису поставлених задач, знання базових понять складання рекурсивних алгоритмів і використання їх при вирішенні інших подібних завдань. Тому,

вдаючись до рекурсії, слід розуміти, яку вигоду можна від неї отримати, і скоротити внесок негативних якостей рекурсії.

Наступним важливим моментом є взаємодія програми, що запускається на виконання, з операційною системою (ОС). Основним завданням ОС є створення для неї процесу і виділення для нього **стека** і «**купи**».

«Купа» використовується для зберігання створюваних в процесі роботи програми динамічних змінних, а стек служить для організації доступу до більшості функцій.

Так, при виконанні функції в стек заносяться її параметри в порядку «справа наліво». Тобто останній аргумент кладеться в стек в першу чергу, за ним кладеться передостанній аргумент, і так далі, поки всі аргументи не виявляться в стеці. Останнім в стек поміщається адреса повернення в програму – це адреса, що слідує за викликом функції в програмі. Після цього управління передається на початок функції. Це забезпечує можливість здійснити рекурсію, тобто зробити повторний виклик тієї ж функції безпосередньо з її тексту або через ланцюжок проміжних викликів.

У програмуванні рекурсія – це фактично визначення частини функції через саму себе, тобто виклик самої себе безпосередньо в своєму тілі або опосередковано через іншу функцію. Типовими рекурсивними задачами є задачі знаходження факторіала числа, обчислення чисел Фібоначчі і т.д.

Розглянемо програму обчислення факторіала числа *N!*, який можна визначити наступним чином:

$$N! = \begin{cases} \text{если } N = 1, \text{то } N! = 1 \\ \text{если } N > 1, \text{то } N! = N * (N-1)! \end{cases}$$

Програма рекурсивного вирішення даної задачі приведена в лістингу, розміщеному далі.

Програма рекурсивного обчислення факторіала

```
// Обчислення факторіала n!

#include <iostream>  // підключаємо бібліотеку вводу-виводу
#include <windows.h> // підключаємо заголовний файл локалізації

using namespace std; // оголошуємо простір імен std

// прототип рекурсивної функції
// тип unsigned long int обраний оскільки його
// значення лежать в інтервалі 0-4 млрд.
unsigned long int factorial (unsigned long int);

int main()  // функція main виконується першою
{
    SetConsoleOutputCP(1251); // локалізуємо вивід
```

```
    // опис даних:
    int n;  // локальна зміння для знаходження факторіала
    cout << "Для обчислення факторіалу n! введіть n = ";
    cin >> n;                // вводимо n

    cout << endl << n << "! = "    // виводимо результат
         << factorial (n) << endl; // обчислень -
    // копія n записується в стек і управління передається
    // на початок функції factorial

    return 0;  // успішне закінчення програми
}              // кінець функції main

// рекурсивна функція знаходження факторіала n!
unsigned long int factorial ( // опис формальних параметрів
         unsigned long int n // число, для якого шукаємо факторіал
){
    // умова повернення з рекурсії
    if ( n == 1 || n == 0 ) // перевіряємо базове або часткове рішення для
    {                       // зупинки процесу рекурсивних викликів
        cout << "Рекурсивний спуск "  // виводимо номер рівня
        << " n = " << n << endl;      // і значення n на цьому кроці
        return 1;      // при n = 1 або n = 0 видаємо результат = 1
    }

    cout << "Рекурсивний спуск "   // виводимо номер рівня
        << " n = " << n << endl;   // і значення n на цьому кроці

    return n*factorial(n-1); // викликаємо саму функцію з аргументом на 1
                             // менше і в кінці повертаємо результат
}
```

```
Для обчислення факторіалу введіть n = 5
Рекурсивний спуск 5
Рекурсивний спуск 4
Рекурсивний спуск 3
Рекурсивний спуск 2
Рекурсивний спуск 1
5! = 120
```

Рис. 3.1. Результат виконання програми

У програмі тип `unsigned long int` обраний тому, що значення факторіала швидко зростає, наприклад, факторіал числа 20 має наступне значення:

20! =2432902008176640000.

Оператори друку додані для виведення проміжних значень, що характеризують хід виконання програми.

Розглянутий процес рекурсивних викликів для 5! виглядає наступним чином (рис. 3.2).

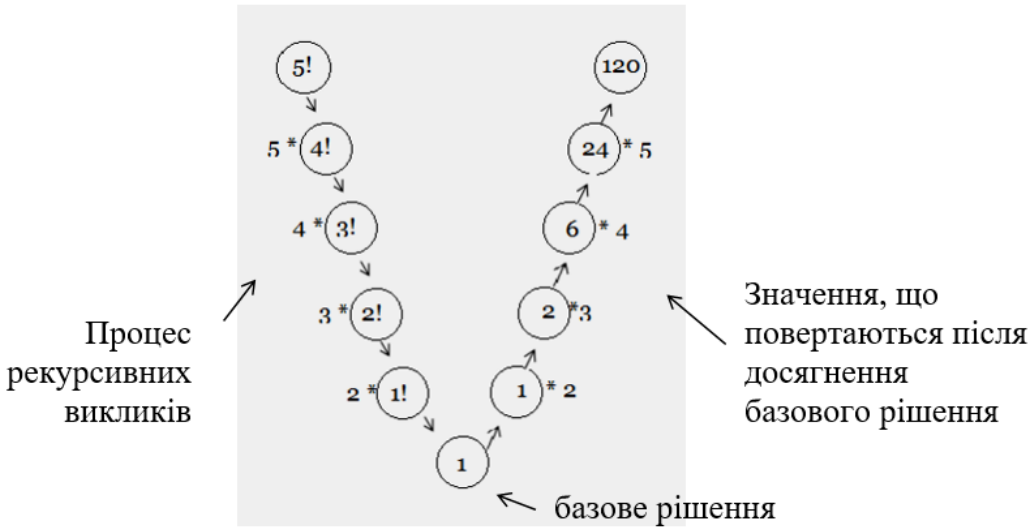

Рис. 3.2. Хід обчислення факторіала

Як правило, звернення до рекурсивної функції розбивається на ряд звернень до даної функції і для кожного з даних звернень створюється своя копія в ОЗУ з усіма локальними змінними, а в стек заносяться відповідні фактичні параметри аж до виходу на базове рішення.

У разі відсутності базового рішення може статися переповнення стека або вихід за межі пам'яті. Породження нових копій рекурсивної програми називається рекурсивним спуском. Максимальна кількість копій рекурсивної функції, які одночасно перебувають у ОЗУ, називається глибиною рекурсії. Завершення роботи рекурсивної функції аж до самої першої називається рекурсивним підйомом.

Завдання до Лабораторної роботи № 3.

1. Розробити програму з метою виконання відповідного варіанту для одного з нижче наведених завдань за допомогою рекурсії.
2. Виконати необхідні розрахунки і **перевірити результат**.

Примітка. При виконанні завдань використовувати нижченаведені співвідношення.

Співвідношення 1.
При розрахунку суми перших *N* чисел натурального ряду.
Обчислення суми *1 + 2 + 3 + ... + N* можна визначити наступним чином:

$$Sum\ (N) = \begin{cases} \text{если } N = 0, \text{то } Sum = 0 \\ \text{если } N > 0, \text{то } Sum = N + Sum(N-1) \end{cases}$$

Співвідношення 2.
Обчислення *N*-го члена послідовності Фібоначчі.

Послідовність Фібоначчі визначається наступним чином: перші два числа дорівнюють 1, а кожне наступне дорівнює сумі двох попередніх (1, 1, 2, 3, 5, ...).

Обчислення N-го члена послідовності Фібоначчі можна визначити наступним чином:

$$Fib\ (N) = \begin{cases} \text{если } N = 1 \text{ или } N = 2, \text{то } Fib = 0 \\ \text{если } N > 2, \text{то } Fib = Fib(N-1) + Fib(N-2) \end{cases}$$

<u>Співвідношення 3.</u>

<u>Розрахунок *n*-го члена арифметичної прогресії, заданої значенням першого члена a_1 і різницею *d*.</u>

Обчислення *n*-члена арифметичної прогресії можна визначити наступним чином:

$$\mathrm{a}_N = \begin{cases} \text{если } N = 1, \text{то } \mathrm{a}_N = a1 \\ \text{если } N > 1, \text{то } \mathrm{a}_N = \mathrm{a}_{N-1} + d \end{cases}$$

<u>Співвідношення 4.</u>

<u>Знаходження суми *N* членів арифметичної прогресії, заданої значеннями першого члена прогресії a_1 і різниці *d*.</u>

Обчислення суми *N* членів арифметичної прогресії можна визначити наступним чином:

$$Sum\ (N, \mathrm{a1}) = \begin{cases} \text{если } N = 0, \text{то } Sum = 0 \\ \text{если } N > 0, \text{то } Sum = \mathrm{a1} + Sum(N-1, \mathrm{a1} + \mathrm{d}) \end{cases}$$

<u>Співвідношення 5.</u>

<u>Піднесення числа *a* до цілого ступеня *n*.</u>

Обчислення *n*-го ступеня числа можна визначити в такий спосіб:

$$\mathrm{a}^n = \begin{cases} \text{если } n = 0, \text{то } \mathrm{a}^n = 1 \\ \text{если } n > 0, \begin{matrix} \text{то } \mathrm{a}^n = a * \mathrm{a}^{n-1} \\ \text{иначе } \mathrm{a}^n = 1/a * \mathrm{a}^{n+1} \end{matrix} \end{cases}$$

<u>Співвідношення 6.</u>

<u>Знаходження найбільшого спільного дільника двох натуральних чисел можна визначити так</u>:

$$NOD\ (a, b\ ) = \begin{cases} \text{если } a = b, \text{то } NOD = a \\ \text{если } a > b, \begin{matrix} \text{то } NOD\ (a - b.b) \\ \text{иначе } NOD\ (a, b - a) \end{matrix} \end{cases}$$

Варіанти завдань:

1. Скласти рекурсивну функцію для обчислення кінцевої суми
$S = 1^2 + 3^2 + \cdots + (2n-1)^2$

2. Скласти рекурсивну функцію для обчислення кінцевої суми
$S = 1^3 + 2^3 + \cdots + n^3$

3. Скласти рекурсивну функцію для обчислення кінцевої суми
$S = 1^2 + 2^2 + \cdots + n^2$

4. Скласти рекурсивну функцію для обчислення кінцевої суми
$S = 2 + 4 + \cdots + 2n$

5. Скласти рекурсивну функцію для обчислення кінцевої суми
$S = 1 + 3 + \cdots + 2n - 1$

6. Скласти рекурсивну функцію для обчислення кінцевої суми
$S = 1 + 2 + 3 + \cdots + n$

7. Скласти рекурсивну функцію для обчислення кінцевої суми
$S = 4 + 8 + 12 + \cdots + 4n$

8. Скласти рекурсивну функцію для обчислення кінцевої суми
$S = 5 + 10 + 15 + \cdots + 5n$

9. Скласти рекурсивну функцію для обчислення добутку
P = 2 *4 *6 *... *2 *n ...

10. Скласти рекурсивну функцію обчислення *n*-го члена арифметичної прогресії: 3, 7, ... і вивести перші 10 членів прогресії.

11. Скласти рекурсивну функцію обчислення *n*-го члена геометричної прогресії: 1, 2, ... і вивести перші 8 членів прогресії.

12. Скласти рекурсивну функцію обчислення *n*-го члена послідовності: $a_1=1$, $a_i=a_{i-1}*i$. Знайти суму 2-го і 5-го членів послідовності.

13. Скласти рекурсивну функцію обчислення n-го члена послідовності: $a_1=0$, $a_i=2*a_{i-1}+i$. Знайти добуток 3-го і 7-го членів послідовності.

14. Скласти рекурсивну функцію знаходження суми *n* членів арифметичної прогресії 1, 3, ... Знайти суму з 5-го по 10-й членів прогресії

15. Скласти рекурсивну процедуру, яка друкує введене натуральне число у вісімковій системі числення

16. Знайти найбільший спільний дільник для чисел A, B, C, використовуючи рекурсивну функцію знаходження найбільшого спільного дільника двох натуральних чисел

17. Скоротити дріб *a/b* (*a*, *b* – введені натуральні числа), використовуючи рекурсивну функцію знаходження найбільшого спільного дільника двох натуральних чисел

18. Скласти рекурсивну функцію обчислення суми цифр довільного натурального числа.

19. Скласти рекурсивну функцію обчислення середнього арифметичного чисел довільного натурального числа.

20. Обчислити $(4^2+2^3)/2^{-2}$, використовуючи рекурсивну функцію піднесення до ступеня

21. Обчислити $(5^3-3^3)/2^{-1}$, використовуючи рекурсивну функцію піднесення до ступеня

22. Скласти рекурсивну функцію, яка обчислює середнє арифметичне елементів одновимірного масиву.

23. Скласти рекурсивний алгоритм знаходження максимального елемента одновимірного масиву.

24. Скласти рекурсивний алгоритм знаходження мінімального елемента одновимірного масиву.

25. Скласти рекурсивний алгоритм знаходження перших N чисел Фібоначчі.

<u>Приклад виконання типового Завдання Лабораторної роботи №3.</u>

Умова:

Скласти рекурсивну функцію для обчислення кінцевої суми

$$S = 1 + \frac{1}{2} - \frac{1}{3} + \frac{1}{4} - \frac{1}{5} \dots \pm \frac{1}{n}$$

Перевірити результат.

```
// Лабораторна робота № 3
// Завдання № 1
// Скласти рекурсивну функцію для обчислення кінцевої суми
// S = 1 + 1/2 – 1/3 ... (+ -) 1 / n. Перевірити результат.

#include <iostream> // підключаємо бібліотеку вводу-виводу
#include <windows.h> // підключаємо заголовний файл локалізації

using namespace std; // оголошуємо простір імен std

float sum (int); // прототип рекурсивної функції обчислення ряду

int main ()     // функція main виконується першою
{
    SetConsoleOutputCP(1251); // локалізуємо вивід

    // опис даних:
    int n;  // кількість членів ряду
    cout << "Введіть кількість членів ряду n = "; // запрошення
    cin >> n;      // вводимо n
    cout << endl;  // переводимо рядок

    cout << "Результат рекурсії sum = " // виводимо результат
         << sum (n) << endl;           // обчислень

    // перевірка обчислення суми членів ряду
    float one,      // змінна для зміни знаку членів ряду
          S = 0.0;  // сума членів ряду, спочатку = 0.0
    for (int i = 1; i <= n; i ++) // підсумовуємо по n
    {
        if ( i == 1 || i% 2 == 0 )  // додатний знак для
            one = +1.0;             // 1-го і парних членів ряду
        else
            one = -1.0;  // для непарних – від’ємний
        S += one / i;    // підсумовуємо члени ряду
```

```
    }
    cout << endl << "Результат перевірки S = " << S << endl;

    return 0;  // успішне закінчення програми
}              // кінець функції main

// рекурсивна функція знаходження суми членів ряду
float sum ( // опис формальних параметрів:
    int n   // кількість членів ряду
){
    float one;  // змінна для зміни знаку членів ряду

    // умова повернення з рекурсії
    if ( n == 1 )  // відомо, що
        return n; // при n = 1 f (n) = 1
    else
        if ( n == 1 || n% 2 == 0 ) // додатний знак для
            one = +1.0;   // 1-го і парних членів ряду
        else
            one = -1.0;   // для непарних – від’ємний

    return sum(n-1) + one / n; // функція викликає саму себе
}
```

```
Введіть кількість членів ряду n = 6

Результат рекурсії sum = 1.38333

Результат перевірки S = 1.38333
```

Рис. 3.3. Результат виконання програми

ЛАБОРАТОРНА РОБОТА № 4

Методи сортування. Сортування в масивах. Основні алгоритми реалізації

Мета роботи: ознайомлення з основними методами сортування, а також дослідження алгоритмів сортування та вивчення способів їх реалізації за допомогою мови програмування С++.

Теоретичні відомості. Методи сортування.

Сортування – це розташування елементів множини даних в певній послідовності: в порядку зростання або зменшення ключів – ознак сортування.

Ключем може бути елемент запису, за яким здійснюється сортування, або будь-яке поєднання пошукових ознак, представлене логічним виразом.

Сортування потрібне для того, щоб, наприклад, забезпечити більш ефективну обробку великих наборів даних або представити людині масиви даних у формі, зручній для аналізу, побудови гістограм розподілу даних і т.д.

Алгоритм сортування – це алгоритм, що описує процес упорядкування елементів в деякому списку. У разі, коли елемент списку має кілька полів, поле, що служить критерієм порядку, називається ключем сортування. На практиці в якості ключа часто виступає число, а в інших полях зберігаються будь-які дані, які не впливають на роботу алгоритму.

За сферою застосування алгоритми сортування класифікуються наступним чином:

Внутрішнє сортування – це алгоритм сортування, який в процесі упорядкування даних використовує тільки оперативну пам'ять (ОЗП) на комп'ютері.

Зовнішнє сортування – це алгоритм сортування, який при упорядкуванні даних використовує зовнішню пам'ять (жорсткі диски). Зовнішнє сортування розроблене для обробки великих списків даних, які не поміщаються в оперативну пам'ять.

Внутрішнє сортування є базовою для будь-якого алгоритму зовнішнього сортування – окремі частини масиву даних упорядковано в оперативній пам'яті і за допомогою спеціального алгоритму зчіплюються в один масив, впорядкований за ключем. Внутрішнє сортування є значно ефективнішим ніж зовнішнє, оскільки на звернення до оперативної пам'яті витрачається набагато менше часу, ніж до носіїв.

Розрізняють такі загальні методи сортування: обміном, вставкою, вибором, злиттям і деякі інші.

Кожен загальний метод сортування формує підгрупу споріднених методів, які мають такі назви:

– методи сортування обміном (стандартний обмін, «бульбашкове», сортування змішуванням, гнома, швидке, гребінцем);

– методи сортування вставками (простими вставками, Шелла, бінарними вставками);

– методи сортування вибором (простим вибором, плавне сортування);

– методи сортування злиттям (простим злиттям, каскадним злиттям) (рис. 4.1).

Рис. 4.1. Методи і алгоритми сортування

<u>Найбільш часто використовуються наступні методи сортування:</u>

– сортування бульбашкою (за зростанням або зменшенням);
– сортування простими вставками (за зростанням або спаданням);
– сортування Шелла (за зростанням або спаданням);
– швидке сортування (за зростанням або спаданням);
– сортування вибором (за зростанням або спаданням);
– сортування злиттям (за зростанням або спаданням).

Розглянемо докладніше алгоритми цих методів сортування.

4.1. Сортування бульбашкою

Ідея методу полягає в наступному: кожен крок сортування полягає в проході по масиву від початку до кінця. По дорозі порівнюються пари сусідніх елементів. Якщо елементи деякої пари знаходяться в неправильному порядку, то їх міняємо місцями. Для реалізації упорядкуємо масив за зростанням. Після нульового проходу по масиву «вгорі» опиниться найбільш «легкий» елемент – звідси аналогія з бульбашкою. Наступний прохід виконується до другого зверху елемента, таким чином, другий за величиною елемент піднімається на правильну позицію. І так далі (рис. 4.2).

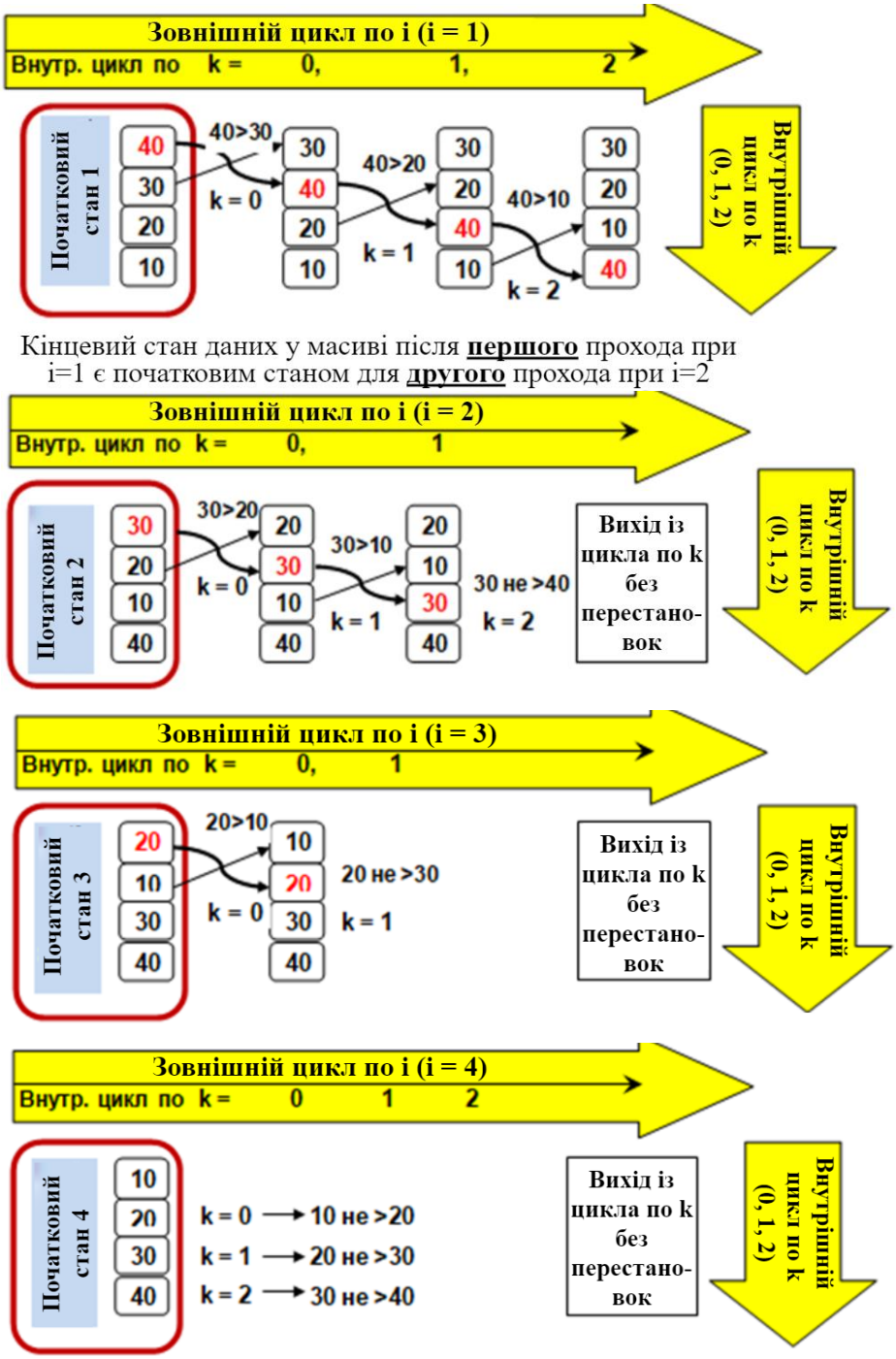

Рис. 4.2. Таблично-графічна схема роботи методу сортування бульбашкою

Розглянемо програмну реалізацію методу сортування бульбашкою.

```
// Програма для дослідження методу сортування бульбашкою

#include <iostream> // підключаємо бібліотеку вводу-виводу
#include <windows.h> // підключаємо заголовний файл локалізації

using namespace std; // оголошуємо простір імен std

void bubbleSort(int [], int);     // прототип функції сортування
void printDiap(int [], int, int); // прототип функції виводу

int main()   // функція main виконується першою
{
    SetConsoleOutputCP(1251); // локалізуємо вивід

    const int n = 4;  // оголошуємо розмір масиву
    // описуємо цілий масив для сортування
    int arr[n] = {40, 30, 20, 10}; // елементи розташовані
                                   // за спаданням

    cout << "Дан масив, ел-ти якого розташовані за спаданням." << endl;
    cout << "Сортуємо за методом бульбашки за зростанням:" << endl;
    bubbleSort(arr, n);  // сортуємо масив

    // перевіряємо вміст масиву arr
    cout << endl << "Результат сортування:  ";
    printDiap(arr, 0, n-1);  // виводимо масив
    cout << endl; // переводимо рядок

    return 0; // успішне закінчення програми
}             // кінець функції main

// функція сортування бульбашкою за зростанням
void bubbleSort( // опис формальних параметрів:
    int a[],    // масив для сортування
    int n       // кількість елементів масиву
){
    int buff;  // змінна для обміну місцями ел-тів масиву

    // цикл по i задає кількість проходів по масиву,
    // а також у внутрішньому циклі по k зменшує
    // кількість порівнянь від початку масиву
    for (int i=1; i<n; i++)
    {
        cout << endl << "Прохід "<< i << endl;
        cout << "Поточний стан масиву:  ";
        printDiap(a, 0, n-1);  // виводимо поточний стан
        cout << "Індекс в циклі по k =   ";

        // в циклі по k проводимо порівняння
```

```
        // сусідніх елементів масиву
        for (int k = 0; k < n-i; k++)
        {
            cout << k << "    ";    // виводимо k
            if ( a[k] > a[k + 1] ) // якщо поточний елемент
            {                      // більший ніж наступний
                buff = a[k];       // поточний – в буфер
                a[k] = a[k + 1];   // наступний – на його місце
                a[k + 1] = buff;   // поточний – на місце
            }                      // наступного
        }
        cout << endl;         // переводимо рядок
        cout << "Результат сортування:  "; // друкуємо
        printDiap(a, 0, n-1); // виводимо стан масиву
    }
}

// функція виводу фрагмента одновимірного масиву
void printDiap ( // опис формальних параметрів:
    int a[],     // масив, що підлягає виводу
    int ii,      // початковий індекс виведення
    int in       // кінцевий індекс виведення
){
    // цикл виведення елементів масиву
    for (int j=ii; j<=in; j++)
        cout << a[j] << "  ";  // виводимо в рядок
    cout << endl;              // переводимо рядок
}
```

```
Дан масив, ел-ти якого розташовані за спаданням.
Сортуємо за методом бульбашки за зростанням:

Прохід 1
Поточний стан масиву:  40  30  20  10
Індекс в циклі по k =   0    1    2
Результат сортування:  30  20  10  40

Прохід 2
Поточний стан масиву:  30  20  10  40
Індекс в циклі по k =   0    1
Результат сортування:  20  10  30  40

Прохід 3
Поточний стан масиву:  20  10  30  40
Індекс в циклі по k =   0
Результат сортування:  10  20  30  40

Результат сортування:  10  20  30  40
```

Рис. 4.3. Результат роботи програми сортування бульбашкою

З рис. 4.3. видно, що **крім руху мінімального елемента вліво відбувається рух максимального вправо!**

За перший прохід максимальний елемент масиву зсувається на три (!) позиції вправо, опиняючись на останньому місці, а мінімальний «піднімається» тільки на одну (!).

За другий прохід найлівіший елемент зсувається вправо на дві позиції, а мінімальний – ще на одну вліво. І, нарешті, на третьому проході відбувається останній обмін.

Примітка.

Стандартне «бульбашкове» сортування може бути модифіковане умовою Айверсона, яка свідчить, що сортування можна припинити достроково, якщо на якомусь етапі в ході порівняння не буде зроблено жодної перестановки.

Фрагмент програми сортування за зростанням бульбашковим сортуванням з виконанням умови Айверсона представлений в лістингу 1.

Лістінг 1.

```
// функція бульбашкового сортування за зростанням
void bubbleIverson ( // опис формальних параметрів:
    int a [],       // масив для сортування
    int n           // кількість елементів масиву
) {
    int buff;  // змінна для обміну місцями ел-тів масиву

    // цикл по i задає кількість проходів по масиву,
    // а також у внутрішньому циклі по k зменшує
    // кількість порівнянь від початку масиву
    for (int i=1; i<n; i++)
    {
        bool ivers = 1; // прапор умови Айверсона піднято
        // проводимо порівняння в циклі по k
        // сусідніх елементів масиву
        for (int k=0; k<n-i; k++)
        {
            if ( a[k] > a[k + 1]) // якщо поточний елемент
            {                     // більше подальшого
                buff = a [k];      // переставляємо
                a [k] = a [k + 1]; // елемент масиву
                a [k + 1] = buff;  // через buff
                ivers = 0;         // опускаємо прапор якщо
            }                      // є перестановка
        }
        // якщо перестановки в циклі по k не було,
        if (ivers)  // то за умовою Айверсона
            return; // виходимо з функції
    }
}
```

На рис. 4.4 представлений результат сортування масиву з чотирьох елементів методом бульбашкового сортування.

```
Дано масив з чотирьох елементів.
Сортуємо методом бульбашки за зростанням:

Прохід 1
Вихідний стан:    20  10  30  40
Індекс в циклі k =  0    1    2
       Результат: 10  20  30  40

Прохід 2
Вихідний стан:    10  20  30  40
Індекс в циклі k =  0    1
       Результат: 10  20  30  40

Прохід 3
Вихідний стан:    10  20  30  40
Індекс в циклі k =  0
       Результат: 10  20  30  40

Кінцевий результат: 10  20  30  40
```

Рис. 4.4. Результат сортування масиву методом бульбашкового сортування

На рис. 4.5. представлений результат сортування тим же методом, але з використанням умови Айверсона.

```
Дано масив з чотирьох елементів.
Сортуємо методом бульбашки за зростанням,
використовуючи умову Айверсона:

Прохід 1
Вихідний стан:    20  10  30  40
Індекс в циклі k =  0    1    2
       Результат: 10  20  30  40

Прохід 2
Вихідний стан:    10  20  30  40
Індекс в циклі k =  0    1

Кінцевий результат: 10  20  30  40
```

Рис. 4.5. Результат сортування масиву методом бульбашкового сортування з використанням умови Айверсона

З наведених результатів видно, що використання умови Айверсона дозволяє скорочувати кількість обчислень. Зауважимо, що друк результатів проводилася також як і в програмі дослідження методу бульбашкового сортування.

4.2. Сортування за методом вставок (Insertion sort)

Ідея методу полягає в наступному:

1. На вхід алгоритму подається послідовність чисел, які називаються ключами.
2. Здійснюється перебір елементів в невідсортованій частині масиву.

3. Кожен елемент вставляється у відсортовану частину масиву на те місце, де він повинен знаходитися.

4. На виході алгоритм повертає відсортовану послідовність ключів.

В процесі виконання алгоритму елементи вхідної послідовності проглядаються по черзі і кожен новий розглянутий елемент переміщується у відповідне місце серед раніше упорядкованих елементів.

Як правило, на початковій стадії процесу сортування, вхідний масив ділиться на 2 частини – відсортовану і невідсортовану.

З невідсортованої частини масиву витягується будь-який елемент. Оскільки інша частина (за визначенням) є відсортованою, то в ній можна досить швидко відшукати потрібне місце для елемента, що розглядається.

Даний елемент вставляється у відповідну позицію відсортованої частини, в результаті чого відсортована частина масиву збільшується, а невідсортована – зменшується. Даний процес завершується тоді, коли в невідсортованій частині масиву завершуються елементи для розгляду.

Тут і далі в якості вхідної послідовності розглядається масив з 6-ти цілих чисел (рис. 4.4.).

Даний масив сортується з метою отримання послідовності ключів, розташованих за зростанням.

Рис. 4.4. Масив, відсортований за методом вставок за зростанням

Нижче наведено таблично-графічне представлення алгоритму сортування вставками (рис. 4.5, 4.6), реалізованого у вигляді програми на мові C++, наведеної далі.

У першому стовпчику таблиці представлені змінні, що визначають хід виконання алгоритму, а також вираз, що впливає на процес перестановки елементів відповідно до виконання алгоритму за методом вставок.

Далі наводиться поетапна зміна ключових змінних: `i, j, key, temp`, а також вмісту масиву `A`, що підлягає сортуванню.

i=	i=1		i=2		i=3			
key=i+1	key=2	key=1	key=3		key=4			
temp=A[key]	temp=8		temp=9		temp=10			
j=i+1	j=2		j=3	j=2	j=4	j=3	j=2	
temp < A[j-1] A[j]:=A[j-1] key := j-1	8<11 A[2]=A[1] key=1		9<11 A[3]=A[2] key=2	9<8 false	10<11 A[4]=A[3] key=3	10<9 false	10<8 false	
		A[key]=temp		A[key]=temp				A[key]=temp

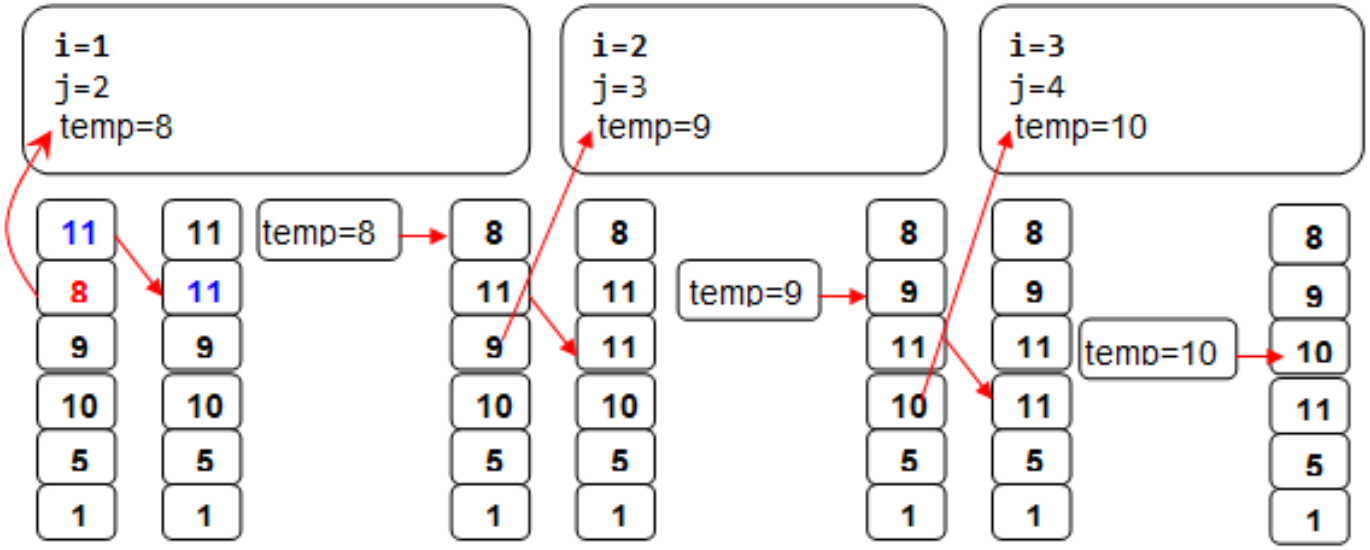

Рис. 4.5. Таблично-графічне представлення початку виконання сортування методом вставок

i	i=4				i=5				
key=i+1	key=5				key=6				
temp=A[key]	temp=5				temp=1				
j=i+1	j=5	j=4	j=3	j=2	j=6	j=5	j=4	j=3	j=2
temp < A[j-1] A[j]:=A[j-1] key := j-1	5<11 A[5]=A[4] key=4	5<10 A[4]=A[3] key=3	5<9 A[3]=A[2] key=2	5<8 A[2]=A[1] key=1	1<11 A[6]=A[5] key=5	1<10 A[5]=A[4] key=5	1<9 A[4]=A[3] key=5	1<8 A[3]=A[2] key=5	1<5 A[2]=A[1] key=5
				A[key]=temp					A[key]=temp

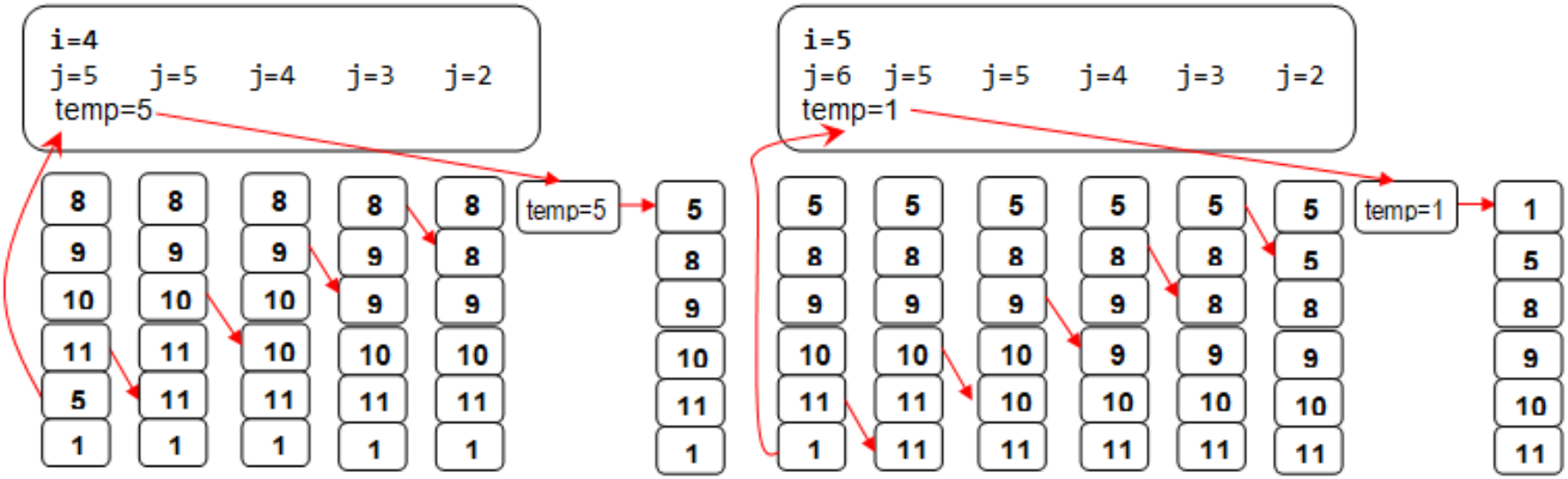

Рис. 4.6. Таблично-графічне представлення етапів завершення сортування методом вставок

Слід особливо відзначити, що поділ масиву на 2 частини в даній реалізації здійснюється просто: першою частиною вважається перший елемент масиву.

Програма реалізації методу сортування вставками на С++

```
// Програма для дослідження методу сортування вставками

#include <iostream> // підключаємо бібліотеку вводу-виводу
#include <windows.h> // підключаємо заголовний файл локалізації

using namespace std; // оголошуємо простір імен std

void insertSort(int [], int);     // прототип функції сортування
void printDiap(int [], int, int); // прототип функції виводу масива

int main()       // функція main виконується першою
{
    SetConsoleOutputCP(1251); // локалізуємо вивід

    const int n = 5; // оголошуємо розмір масиву
    // описуємо цілий масив для сортування
    int arr[n] = {11, 9, 10, 5, 1};  // елементи масиву

    cout << "     Сортування вставками: ";
    printDiap(arr, 0, n-1);  // виводимо масив
    cout << endl;            // переводимо рядок

    cout << "Сортуємо за зростанням: ";
    insertSort(arr, n); // сортуємо масив

    // перевіряємо вміст масиву arr
    cout << endl << "      Відсортований масив: ";
    printDiap(arr, 0, n-1); // виводимо масив

    return 0;  // успішне закінчення програми
}              // кінець функції main

// функція сортування вставками за зростанням
void insertSort( // опис формальних параметрів:
    int a[],     // масив для сортування
    int n        // кількість елементів масиву
){
    int buff,  // змінна для зберігання елемента сортованого масиву
        iPrev; // індекс попереднього елемента

    // цикл алгоритму починається не з 0-го, а з 1-го елемента,
    // оскільки масив, що складається з одного елемента є
    // відсортованим і вже відносно нього ми будемо вставляти інші
    for (int i=1; i<n; i++)
    {
        buff = a[i]; // ініціалізуємо змінну поточним значенням елемента
```

```
        iPrev = i-1; // запам'ятовуємо індекс попереднього елемента масиву

        // друкуємо поточний стан масиву
        cout << endl << "Крок " << i << " - порівнюємо ел-ти: ";
        printDiap(a, 0, i); // виводимо масив

        // вкладений цикл шукає місце для вставки, змінюючи
        // місцями елементи a[iPrev + 1] і a[iPrev]
        while (iPrev >= 0 && a[iPrev] > buff) // поки індекс більше або
                                              // дорівнює 0 і попередній
        {                                 // елемент масиву більше поточного
            a[iPrev + 1] = a[iPrev];      // переставляємо елементи масиву
            a[iPrev] = buff;              // через buff
            iPrev--;                      // декрементуємо індекс iPrev

            // друкуємо поточний стан масиву
            cout << " Перевіряємо і вставляємо: ";
            printDiap(a, 0, i);
        }
    }
}

// функція виведення фрагмента одновимірного масиву
void printDiap( // опис формальних параметрів:
    int a[],    // масив для виведення
    int ii,     // початковий індекс виведення
    int in      // кінцевий індекс виведення
){
    // цикл виведення елементів масиву
    for (int j=ii; j<=in; j++)
        cout << a[j] << "  ";  // виводимо в рядок
    cout << endl;              // переводимо рядок
}
```

```
     Сортування вставками: 11  9  10  5  1

Сортуємо за зростанням:
Крок 1 - порівнюємо ел-ти: 11  9
 Перевіряємо і вставляємо: 9  11

Крок 2 - порівнюємо ел-ти: 9  11  10
 Перевіряємо і вставляємо: 9  10  11

Крок 3 - порівнюємо ел-ти: 9  10  11  5
 Перевіряємо і вставляємо: 9  10  5  11
 Перевіряємо і вставляємо: 9  5  10  11
 Перевіряємо і вставляємо: 5  9  10  11

Крок 4 - порівнюємо ел-ти: 5  9  10  11  1
 Перевіряємо і вставляємо: 5  9  10  1  11
 Перевіряємо і вставляємо: 5  9  1  10  11
 Перевіряємо і вставляємо: 5  1  9  10  11
 Перевіряємо і вставляємо: 1  5  9  10  11

      Відсортований масив: 1  5  9  10  11
```

Рис. 4.7. Результат роботи програми сортування вставками

4.3. Сортування за методом Шелла

Ідея методу полягає в наступному.

1. Порівнюються не тільки ті елементи, що, знаходяться поруч, але й елементи на певній відстані один від одного.

2. Вибір відстані здійснюється різними способами, але спочатку це відстань вважається рівною **d** або **N/2**, де **N** – загальне число елементів.

На першому кроці кожна група включає в себе два елементи, розташовані один від одного на відстані **N/2**; вони порівнюються між собою, і, в разі необхідності, міняються місцями. На наступних кроках також відбуваються перевірка і обмін, але відстань **d** скорочується на **d/2**, і кількість груп, відповідно, зменшується. Поступово відстань між елементами зменшується, і при зменшенні відстані до значення **d=1** прохід по масиву відбувається в останній раз.

Цей алгоритм може розглядатися і як узагальнення сортування бульбашкою, так і сортування вставками.

Тут в якості вхідної послідовності розглядається масив з 6-ти цілих чисел (рис. 4.8.).

Даний масив сортується з метою отримання послідовності ключів, розташованих за зростанням.

Рис. 4.8. Масив, сортований за методом Шелла за зростанням

Нижче наведено таблично-графічне представлення алгоритму сортування за методом Шелла (рис. 4.9, 4.10), реалізованого у вигляді програми на мові C++, наведеної далі.

У першому стовпчику таблиці представлені змінні, що визначають хід виконання алгоритму, а також вираз, що впливає на процес перестановки елементів відповідно до виконання алгоритму за методом Шелла.

Далі наводиться поетапне зміна вмісту ключових змінних: `n`, `d`, `i`, `j`, `temp`, а також вмісту масиву `A`, що підлягає сортуванню.

№ кроку	1 →	2 →	3 →	4 →	5
n=6	n-d=3				
d=6	d=3				
i=	i=1	i=2		i=3	
j=i	j=1	j=2	j=1	j=3	j=2
j+d	j+d=4	j+d=5	j+d=4	j+d=6	j+d=5
A[j] > A[j+d]	A[1] > A[4]	A[2] > A[5]	A[1] > A[4]	A[3] > A[6]	A[2] > A[5]
	11>10	8>5	10>11	9>1	5>8
	true	true	false	true	false
temp = A[j] A[j] =A[j+d] A[j+d] = temp	A[1] = A[4] A[4] = A[1]	A[2] = A[5] A[5] = A[2]	Обміну немає	A[3] = A[6] A[6] = A[3]	Обміну немає
	j=j-1=0	j=j-1=1		j=j-1=2	

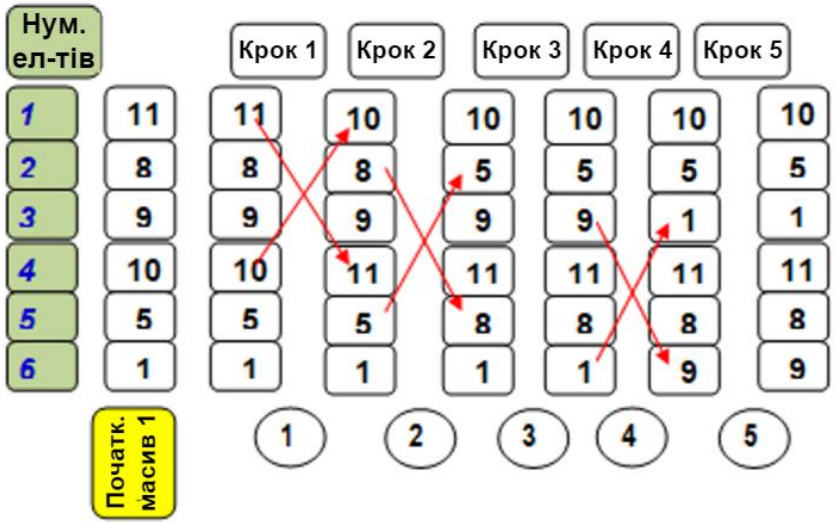

Рис. 4.9. Таблично-графічне представлення початкових етапів виконання методу Шелла

№ кроку	6 →	7 →	8 →	9 →	10 →	11 →	12 →	13
n=6	n-d=5							
d=6	d=1							
i=	i=1	i=2		i=3	i=4			i=5
j=i	j=1	j=2	j=j-1=1	j=3	j=4	j=j-1=3	j=j-1=2	j=5
j+d	j+d=2	j+d=3	j+d=2	j+d=4	j+d=5	j+d=4	j+d=3	j+d=6
A[j] > A[j+d]	A[1] > A[2]	A[2] > A[3]	A[1] > A[2]	A[3] > A[4]	A[4] > A[5]	A[3] > A[4]	A[2] > A[3]	A[5] > A[6]
	10>5	10>1	5>1	10>11	11>8	9>8	5>8	11>10
	true	true	true	false	true	true	false	true
temp = A[j] A[j] =A[j+d] A[j+d] = temp	A[1] = A[2] A[2] = A[1]	A[2] = A[3] A[3] = A[2]	A[1] = A[2] A[2] = A[`]	Обміну немає	A[4] = A[5] A[5] = A[4]	A[3] = A[4] A[4] = A[3]	Обміну немає	A[5] = A[6] A[6] = A[5]
	j=j-1=0	j=j-1=1	j=j-1=0		j=j-1=3	j=j-1=2		j=j-1=4

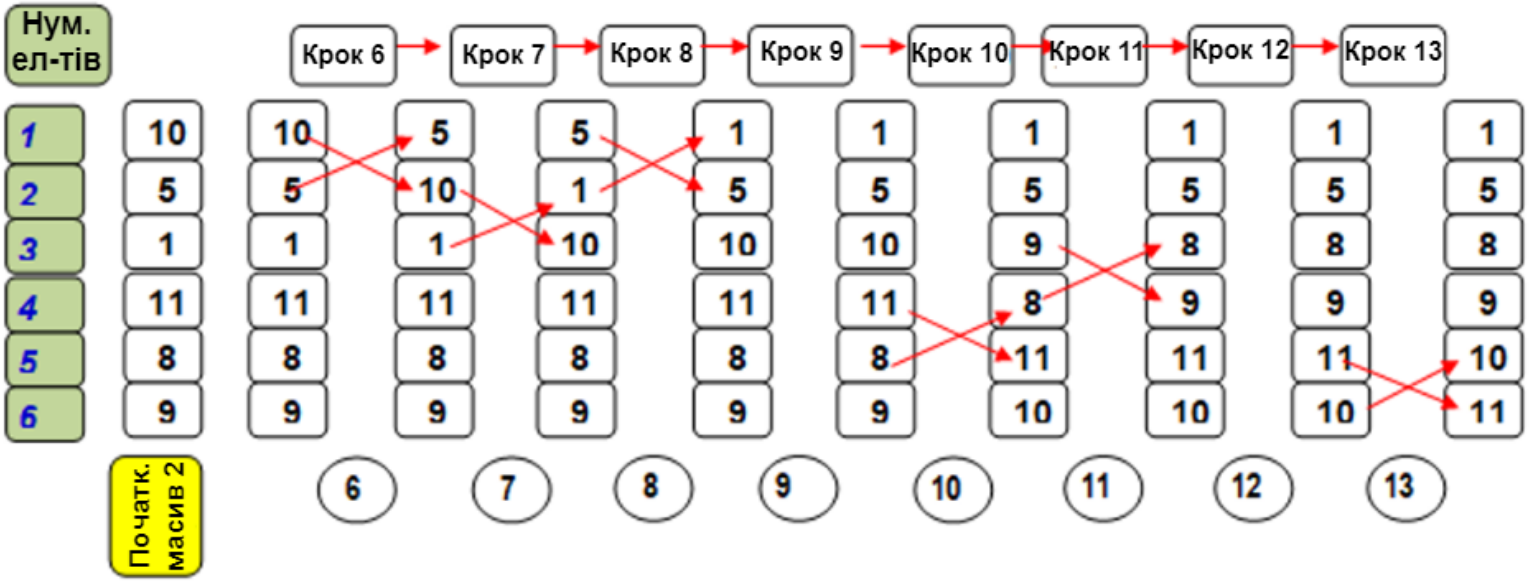

Рис. 4.10. Завершення виконання алгоритму за методом Шелла

Програма реалізації сортування Шелла на мові С++

```
// Програма для дослідження методу сортування Шелла

#include <iostream> // підключаємо бібліотеку вводу-виводу
#include <windows.h> // підключаємо заголовний файл локалізації

using namespace std; // оголошуємо простір імен std

void insertShell(int [], int);    // прототип функції сортування
void printDiap(int [], int, int); // прототип функції виводу масива

int main()    // функція main виконується першою
{
    SetConsoleOutputCP(1251); // локалізуємо вивід

    const int n = 5; // оголошуємо розмір масиву
    // описуємо цілий масив для сортування
    int arr[n] = {11, 9, 10, 5, 1};  // елементи масиву

    cout << "        Сортування Шелла: ";
    printDiap(arr, 0, n-1); // виводимо масив
    cout << endl;           // переводимо рядок

    cout << "Сортуємо за зростанням: ";
    insertShell(arr, n); // сортуємо масив

    // перевіряємо вміст масиву arr
    cout << endl << "     Відсортований масив: ";
    printDiap(arr, 0, n-1);  // виводимо масив

    return 0;   // успішне закінчення програми
}               // кінець функції main

// функція сортування Шелла за зростанням
void insertShell ( // опис формальних параметрів:
    int a[],       // масив для сортування
    int n          // кількість елементів масиву
){
    int buff; // змінна для зберігання елемента масиву

    // вибираємо крок сортування step, для якого Шелл запропонував
    // послідовність n/2, n/4, n/8 ..., де n – кількість
    // елементів в сортованому масиві
    int step = n/2; // ініціалізуємо крок

    // рухаємося, поки крок не дорівнює 0
    while ( step > 0 )
    {
        cout << endl << "step = " << step << endl; // виводимо step
```

```
        // йдемо, починаючи з i-го елемента
        for (int i=0; i<(n-step); i++)
        {
            int j = i; // запам'ятовуємо поточний елемент

            // поки не прийшли до початку масиву
            // та поки елемент, що розглядається, є більшим
            // ніж елемент, що знаходиться на відстані кроку
            while (j >= 0 && a[j] > a[j + step])
            {
                buff = a[j];        // зберігаємо поточний елемент
                a[j] = a[j + step]; // змінюємо на менший
                a[j + step] = buff; // а менший – на поточний
                j--;                // рухаємося до початку
            }
            // виводимо поточний стан
            cout << "Перевіряємо і вставляємо: ";
            printDiap(a, 0, i);
        }
        step = step/2;  // зменшуємо крок у 2 рази
    }
}

// функція виведення фрагмента одновимірного масиву
void printDiap ( // опис формальних параметрів:
    int a[],     // масив, що підлягає виводу
    int ii,      // початковий індекс виведення
    int in       // кінцевий індекс виведення
){
    // цикл виведення елементів масиву
    for (int j=ii; j<=in; j++)
        cout << a[j] << "  ";  // виводимо в рядок
    cout << endl;            // переводимо рядок
    }
```

```
        Сортування Шелла: 11  9  10  5  1

Сортуємо за зростанням:
step = 2
Перевіряємо і вставляємо: 10
Перевіряємо і вставляємо: 10  5
Перевіряємо і вставляємо: 10  5  1

step = 1
Перевіряємо і вставляємо: 5
Перевіряємо і вставляємо: 1  5
Перевіряємо і вставляємо: 1  5  9
Перевіряємо і вставляємо: 1  5  9  10

     Відсортований масив: 1  5  9  10  11
```

Рис. 4.11. Результат роботи програми сортування Шелла

4.4. Метод швидкого сортування Хоара (quicksort)

Даний метод також називається сортуванням Хоара (за іменем його розробника Чарльза Хоара, який створив цей метод у 1960 р).

Ідея методу полягає в наступному.

1. На першому етапі у вхідному масиві вибирається елемент, що називається опорним. Це може бути будь-який елемент з вхідної послідовності ключів. Однак, від вибору опорного елемента коректність алгоритму не залежить, але в окремих випадках може сильно залежати його ефективність.

2. Всі інші елементи порівнюються з опорним і переставляються в масиві так, щоб розбити його на три безперервних відрізки, що слідують один за одним: «елементи, менші за опорний», «дорівнюють опорному» і «більші за опорний».

3. Для відрізків «менших» і «більших» значень виконати рекурсивно ту ж послідовність операцій, якщо довжина відрізка більше одиниці.

На практиці масив зазвичай ділять не на три, а на дві частини: наприклад, «менші за опорний» та «дорівнюють опорному і більші від нього»; такий підхід в загальному випадку є більш ефективним, оскільки спрощує алгоритм поділу.

Тут в якості вхідної послідовності розглядається масив з 6-ти цілих чисел (рис. 4.12).

Даний масив сортується з метою отримання послідовності ключів, розташованих за зростанням.

Рис. 4.12. Масив, сортований за методом Хоара за зростанням

Нижче наведено таблично-графічне представлення алгоритму методу швидкого сортування (рис. 4.13), реалізованого у вигляді програми на мові C++, представленої далі.

У першому стовпчику таблиці представлені змінні, що визначають хід виконання алгоритму, а також вирази, що впливають на процес перестановки елементів відповідно до виконання алгоритму за методом вставок.

Далі наводиться поетапне зміна вмісту ключових змінних: `first, last, mid, f, t`, а також вмісту сортованого масиву `A`.

Крім того, в таблиці для кращого розуміння перетворень, що відбуваються, вказані границі виконання тіла циклів `repeat` і `while`, а також процес виконання функцій `ink(f++)` і `dec(t--)`.

№ кроку	❶		❷	quicksort (A, 1, 4)	❸	quicksort (A, 1, 3)
first=1	f=1			f=1		f=1
last=6	t=6			t=4		t=3
mid = A[(f + t) div 2]	mid=A[3] 9			mid=A[2] 8		mid=A[2] 5
repeat						
while						
inc	f=1	f=2	f=3	f=1		f=1
	A[f] < 9	A[f] < 9	A[f] < 9	A[f] < 8		A[f] < 5
	A[1] < 9	A[2] < 9	A[3] < 9	A[1] < 8		A[1] < 5
	11<9	8<9; f=3	9<9 f=3	1<8; f=2		1<5; f=2
	false		false	8<8; f=2	f=2	5<5
while						
dec	t=6	t=5	t=5	t=4	t=3	t=3
	A[t] > 9		A[t] > 9	A[t] > 8	A[t] > 8	A[t] > 5
	A[6] >9		A[5] > 9	A[4] >8	A[3] >8	A[3] >5
	1 > 9		5>9	10>8; t=3	5>8	5>5
	false		false		false	false
if f<=t	1<=6		3<=5	1<=6	2<=3	2<=3
temp = A[f] A[f] =A[t] A[t] =temp	A[1] = A[6] A[6] = A[1]		A[3] = A[5] A[5] = A[3]		A[3] = A[2] A[2] = A[3]	A[3] = A[2] A[2] = A[3]
inc	f=2		f=4		f=3	
dec	t=5		t=4		t=3	
first < t			1 < 4		1 < 3	
f < last						

т.д.

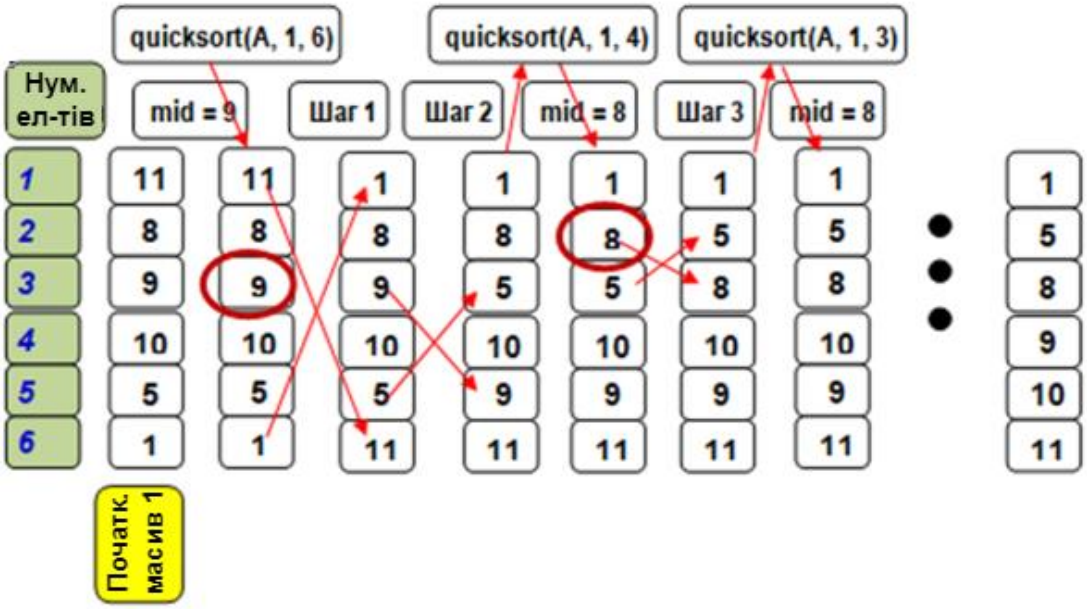

Рис. 4.13. Хід виконання методу швидкого сортування

Програма методу швидкого сортування на мові С++

```
// Програма для дослідження методу швидкого сортування Хоара

#include <iostream> // підключаємо бібліотеку вводу-виводу
#include <windows.h> // підключаємо заголовний файл локалізації

using namespace std; // оголошуємо простір імен std

int partition (int[], int, int); // прототип функції визначення
                                 // індексу опорного елемента
void quickSort (int[], int, int); // прототип функції швидкого сортування
void printDiap (int[], int, int); // прототип функції виведення масиву

int main()     // функція main виконується першою
{
    SetConsoleOutputCP(1251); // локалізуємо вивід

    const int n = 5;  // оголошуємо розмір масиву
    // описуємо цілий масив для сортування
    int arr[n] = {11, 9, 10, 5, 1};  // елементи масиву

    cout << "Швидке сортування: ";
    printDiap(arr, 0, n-1);  // виводимо масив
    cout << endl;            // переводимо рядок

    cout << "Сортуємо за зростанням: " << endl;
    quickSort(arr, 0, n-1);  // сортуємо масив

    // перевіряємо вміст масиву arr
    cout << endl << "Відсортований масив: ";
    printDiap(arr, 0, n-1);  // виводимо масив

    return 0;  // успішне закінчення програми
}              // кінець функції main

// функція швидкого сортування за зростанням
void quickSort ( // опис формальних параметрів:
    int a[],     // масив для сортування
    int start,   // початковий індекс масиву
    int end      // кінцевий індекс масиву
){
    // умова закінчення сортування
    if ( start >= end )
    {
        return;  // сортування закінчено
    }
    // отримуємо індекс опорного елемента
    int pivot = partition(a, start, end);

    // звертаємося рекурсивно:
```

```
    quickSort(a, start, pivot-1); // сортуємо лівий підмасив
    quickSort(a, pivot+1, end);   // сортуємо правий підмасив
}

// функція визначення індексу опорного елемента
int partition ( // опис формальних параметрів:
    int a[],    // масив для сортування
    int start,  // початковий індекс сортування
    int end     // кінцевий індекс сортування
){
    int buff;  // змінна для зберігання елемента сортованого масиву
    int marker = start;  // ділимо лівий і правий підмасиви

    // рухаємося від початкового до кінцевого індексу
    for (int i=start; i<=end; i++)
    {
        if ( a[i] < a[end] )  // a[end] є опорним елементом
        {
            buff = a[marker]; // міняємо місцями
            a[marker] = a[i]; // відповідні
            a[i] = buff;      // елементи
            marker += 1;     // збільшуємо індекс
        }
    }
    // розміщуємо опорний елемент a[end]
    // між лівим і правим підмасивами
    buff = a[marker];   // міняємо місцями
    a[marker] = a[end]; // відповідні
    a[end] = buff;      // елементи

    // роздруковуємо підмасиви
    cout << "Лівий підмасив: ";
    printDiap(a, 0, marker);       // виводимо лівий підмасив
    cout << "Правий підмасив: ";
    printDiap(a, marker + 1, end); // виводимо правий підмасив
    cout << endl;

    return marker; // повертаємо індекс опорного елемента
}

// функція виведення фрагмента одновимірного масиву
void printDiap ( // опис формальних параметрів:
    int a[],     // масив, що підлягає виводу
    int ii,      // початковий індекс виведення
    int in       // кінцевий індекс виведення
){
    // цикл виведення елементів масиву
    for (int j=ii; j<=in; j++)
        cout << a[j] << "  ";  // виводимо в рядок
    cout << endl;              // переводимо рядок
}
```

```
  Швидке сортування: 11  9  10  5  1

Сортуємо за зростанням:
     Лівий підмасив: 1
    Правий підмасив: 9  10  5  11

     Лівий підмасив: 1  9  10  5  11
    Правий підмасив:

     Лівий підмасив: 1  5
    Правий підмасив: 10  9

     Лівий підмасив: 1  5  9
    Правий підмасив: 10


Відсортований масив: 1  5  9  10  11
```

Рис. 4.14. Результат роботи програми швидкого сортування Хоара

4.5. Сортування за методом злиття (mergesort)

Ідея методу полягає в наступному.

1. Масив рекурсивно розбивається навпіл, і кожна з половин ділиться до тих пір, поки розмір чергового підмасиву не стане дорівнювати одиниці.

2. Далі виконується операція алгоритму, що називається злиттям. Два одиничних масиви зливаються в загальний результуючий масив, при цьому з кожного вибирається менший елемент (сортування за зростанням) і записується у вільну ліву клітинку результуючого масиву. Після чого з двох результуючих масивів збирається третій загальний відсортований масив, і так далі. У разі, якщо один з масивів закінчитися, елементи іншого дописують в масив, що збирається.

3. В кінці операції злиття, елементи перезаписуються з результуючого масиву в початковий.

Даний метод широко використовується для впорядкування списків або інших структур даних, доступ до елементів яких можна отримати тільки послідовно, як, наприклад, потоки.

Таблично-графічне представлення виконання сортування методом злиття для цілочисельного масиву з п'яти елементів наведено в таблиці 4.1.

Таблиця 4.1.

Приклад сортування масиву з використанням методу злиття

Початковий масив	11	9	10	5	1
Перша ітерація	11	9	10	5	1
Друга ітерація	11	9	10	5	1
Третя ітерація	11	9	10	5	1
Четверта ітерація	9	11	10	1	5
П'ята ітерація	9	10	11	1	5
Шоста ітерація	1	5	9	10	11
Відсортований масив	1	5	9	10	11

Програма реалізації методу сортування злиттям на мові C++

```
// Програма для дослідження методу сортування злиттям

#include <iostream> // підключаємо бібліотеку вводу-виводу
#include <windows.h> // підключаємо заголовний файл локалізації

using namespace std; // оголошуємо простір імен std

void mergeSort(int [], int, int);  // прототип функції сортування злиттям
void merge(int [], int, int, int); // прототип функції злиття масивів
void printDiap(int [], int, int);  // прототип функції виведення масиву

int main()   // функція main виконується першою
{
    SetConsoleOutputCP(1251); // локалізуємо вивід

    const int n = 5; // оголошуємо розмір масиву
    // описуємо цілий масив для сортування
    int arr[n] = {11, 9, 10, 5, 1};  // елементи масиву

    cout << " Сортування злиттям: ";
    printDiap(arr, 0, n-1);  // виводимо масив
```

```
    cout << endl;            // переводимо рядок

    cout << "Сортуємо за зростанням: " << endl;
    mergeSort(arr, 0, n-1);  // сортуємо масив

    // перевіряємо вміст масиву arr
    cout << endl << "Відсортований масив: ";
    printDiap(arr, 0, n-1);  // виводимо масив

    return 0;  // успішне закінчення програми
}              // кінець функції main

// функція сортування злиттям за зростанням
void mergeSort ( // опис формальних параметрів:
    int a[],     // масив для сортування
    int l,       // початковий індекс сортування
    int r        // кінцевий індекс сортування
){
    int marker;  // індекс опорного елемента

    // умова закінчення сортування
    if ( l < r )
    {
        marker = l + (r-l) / 2;  // обчислюємо індекс опорного елемента

        // звертаємося рекурсивно:
        mergeSort(a, l, marker);    // сортуємо лівий підмасив
        mergeSort(a, marker+1, r);  // сортуємо правий підмасив

        // роздруковуємо підмасиви
        cout << "     Лівий підмасив: ";
        printDiap(a, l, marker);     // виводимо лівий підмасив
        cout << "    Правий підмасив: ";
        printDiap(a, marker + 1, r); // виводимо правий підмасив
        cout << endl;

        merge(a, l, marker, r);  // здійснюємо злиття
    }
}

// функція злиття масивів
void merge ( // опис формальних параметрів:
    int a[], // масив для сортування
    int l,   // початковий індекс сортування
    int m,   // опорний індекс сортування
    int r    // кінцевий індекс сортування
){
    int i, j, k, // індекси для роботи з масивами
        nl,      // розмір лівого підмасива
        nr;      // розмір правого підмасива
```

```
    nl = m-l + 1; // обчислюємо розмір лівого підмасива
    nr = r-m;    // обчислюємо розмір правого підмасива

    // оголошуємо тимчасові масиви
    int larr[nl], // лівий підмасив
        rarr[nr]; // правий підмасив

    // заповнюємо підмасиви:
    for (i=0; i<nl; i++)
        larr[i] = a[l + i];  // лівий підмасив
    for (j=0; j<nr; j++)
        rarr[j] = a[m+1+j];  // правий підмасив

    // готуємо індекси до злиття:
    i = 0;
    j = 0;
    k = l;

 // здійснюємо злиття тимчасових масивів з вихідним
    while ( i < nl && j < nr )
    {
        if ( larr[i] <= rarr[j] )
        {
            a[k] = larr[i];
            i++;
        }
        else
        {
            a[k] = rarr[j];
            j++;
        }
        k++;
    }
    while ( i < nl )
    {
        a[k] = larr[i]; // додатковий елемент в лівому масиві
        i++;
        k++;
    }
    while ( j < nr )
    {
        a[k] = rarr[j]; // додатковий елемент в правому масиві
        j++;
        k++;
    }
}

// функція виведення фрагмента одновимірного масиву
void printDiap ( // опис формальних параметрів:
    int a[],     // масив, що підлягає виводу
    int ii,      // початковий індекс виведення
```

```
    int in       // кінцевий індекс виведення
){
    // цикл виведення елементів масиву
    for (int j=ii; j<=in; j++)
        cout << a[j] << "  ";  // виводимо в рядок
    cout << endl;              // переводимо рядок
}
```

```
Сортування злиттям: 11  9  10  5  1

Сортуємо за зростанням:
     Лівий підмасив: 11
    Правий підмасив: 9

     Лівий підмасив: 9  11
    Правий підмасив: 10

     Лівий підмасив: 5
    Правий підмасив: 1

     Лівий підмасив: 9  10  11
    Правий підмасив: 1  5


Відсортований масив: 1  5  9  10  11
```

Рис. 4.15. Результат роботи програми сортування злиттям

Завдання до Лабораторної роботи №4 для одновимірних і двовимірних масивів

Загальне завдання:

1. Заповнити зазначені в заданих варіантах масиви цілочисельними випадковими числами в діапазоні (10-100).
2. Відсортувати масиви зазначеними методами.

Завдання за варіантом:

1. Відсортувати елементи одновимірного і двовимірного динамічних масивів (сортування бульбашкою і сортування вибором) за зростанням (розмірність масивів: 22, 15x14).
2. Відсортувати елементи одновимірного і двовимірного динамічних масивів (сортування Шелла і сортування злиттям) за спаданням (розмірність масивів: 25, 7x12).
3. Відсортувати елементи одновимірного і двовимірного динамічних масивів (сортування простими вставками і швидке сортування) за зростанням (розмірність масивів: 17, 12x10).
4. Відсортувати елементи одновимірного і двовимірного динамічних масивів (сортування Шелла і сортування злиттям) за спаданням (розмірність масивів: 23, 9x15).
5. Відсортувати другу половину одновимірного масиву за спаданням (сортування простими вставками і швидке сортування) (розмірність масиву: 76).

6. Відсортувати другу половину одновимірного масиву за зростанням (сортування Шелла і сортування злиттям) (розмірність масиву: 82).

7. Відсортувати першу половину одновимірного масиву за зростанням (сортування бульбашкою і сортування вибором) (розмірність масиву: 68).

8. Відсортувати другу половину одновимірного масиву за спаданням (сортування Шелла і сортування злиттям) (розмірність масиву: 92).

9. Відсортувати другу половину одновимірного масиву за спаданням (сортування простими вставками і швидке сортування) (розмірність масиву: 74).

10. Відсортувати першу половину одновимірного масиву за зростанням (сортування бульбашкою і сортування вибором) (розмірність масиву: 96).

11. Відсортувати другу половину одновимірного масиву за зростанням (сортування Шелла і сортування злиттям) (розмірність масиву: 82).

12. Відсортувати першу половину одновимірного масиву за спаданням (сортування простими вставками і швидке сортування) (розмірність масиву: 78).

13. Відсортувати другу половину одновимірного масиву за спаданням (сортування бульбашкою і сортування вибором) (розмірність масиву: 88).

14. Відсортувати елементи одновимірного масиву (сортування Шелла і сортування злиттям) за зростанням (розмірність масиву: 72).

15. Відсортувати елементи одновимірного масиву (сортування простими вставками і швидке сортування) за спаданням (розмірність масиву: 94).

16. Відсортувати елементи одновимірного масиву, що стоять на парних місцях, (сортування бульбашкою і сортування вибором) за спаданням (розмірність масиву: 76).

17. Відсортувати елементи одновимірного масиву, що стоять на непарних місцях, (сортування Шелла і сортування злиттям) за зростанням (розмірність масиву: 86).

18. Відсортувати елементи одновимірного масиву, що стоять на парних місцях, (сортування простими вставками і швидке сортування) за спаданням (розмірність масиву: 78).

19. Відсортувати елементи одновимірного масиву, що стоять на непарних місцях, (сортування бульбашкою і сортування вибором) за зростанням (розмірність масиву: 92).

20. Відсортувати елементи одновимірного масиву, що стоять на парних місцях, (сортування Шелла і сортування злиттям) за спаданням (розмірність масиву: 88).

21. Відсортувати елементи одновимірного масиву, що стоять на непарних місцях, (сортування простими вставками і швидке сортування) за зростанням (розмірність масиву: 76).

22. Відсортувати елементи одновимірного масиву, що стоять на парних місцях, (сортування бульбашкою і сортування вибором) за спаданням (розмірність масиву: 94).

23. Відсортувати елементи одновимірного масиву, що стоять на непарних місцях, (сортування Шелла і сортування злиттям) за зростанням (розмірність масиву: 78).

24. Відсортувати другу половину одновимірного масиву за спаданням (сортування простими вставками і швидке сортування) (розмірність масиву: 86).

25. Відсортувати другу половину одновимірного масиву за зростанням (сортування Шелла і сортування злиттям) (розмірність масиву: 98).

ЛАБОРАТОРНА РОБОТА № 5

Методи пошуку. Пошук в масивах. Основні алгоритми реалізації

Мета роботи: ознайомлення з основними методами пошуку, а також вивчення способів їх реалізації за допомогою мови програмування С++.

Теоретичні відомості. Методи пошуку.

Основним завданням пошуку є наступне. Нехай задано множину ключів $\{k_1, k_2, k_3, ..., k_n\}$.

Необхідно відшукати в даній множині ключ k_i. Пошук може бути завершений в двох випадках:

– ключ у множині відсутній;
– ключ у множині знайдений.

Алгоритми пошуку можна розбити на наступні групи (рис. 5.1).

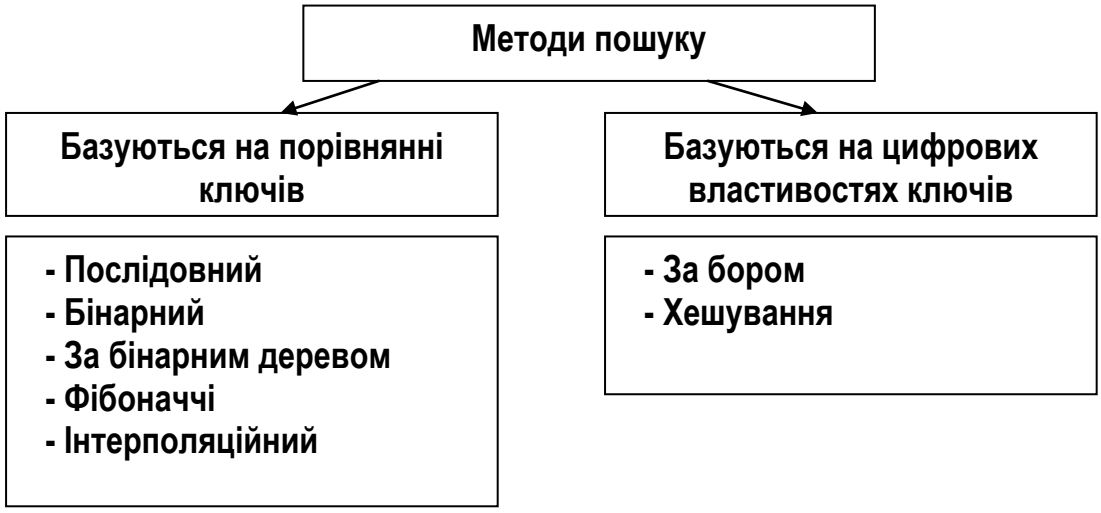

Рис. 5.1. Методи пошуку

Лінійний, послідовний **пошук** – алгоритм знаходження заданого значення довільної функції на деякому відрізку. Даний алгоритм є найпростішим алгоритмом **пошуку** і, на відміну, наприклад, від бінарного **пошуку**, не накладає ніяких обмежень на функцію і має найпростішу реалізацію.

Послідовний пошук виконується шляхом перебору всіх елементів один за іншим. Перевіряється кожен елемент, і якщо збіг знайдено, повертається цей конкретний елемент і значення номера його розташування.

5.1. Двійковий (бінарний) пошук

Двійковий (бінарний) пошук (також відомий як метод поділу навпіл і дихотомія) – класичний алгоритм пошуку елемента у **відсортованому масиві (векторі)**, який використовує розбиття масиву на половини. Використовується в інформатиці, обчислювальній математиці та математичному програмуванні.

Окремим випадком двійкового пошуку є метод бісекції, який застосовується для пошуку коренів заданої безперервної функції на заданому відрізку.

Алгоритм двійкового пошуку елемента у відсортованому масиві (за зростанням або спаданням) може бути описаний таким чином.

1. Визначення значення елемента шляхом знаходження його в середині структури даних. Отримане значення порівнюється з ключем.
2. Якщо ключ є меншим за значення середини, то пошук здійснюється в першій (лівій) половині елементів, інакше – у другій (правій).
3. Пошук зводиться до того, що знову визначається значення серединного елемента в обраній половині і порівнюється з ключем.
4. Процес триває до тих пір, поки не буде знайдений елемент зі значенням ключа або не стане порожнім інтервал для пошуку.

Алгоритм може бути визначений в рекурсивній та нерекурсивній формах.

Розглянемо докладніше обчислювальну складову методу двійкового пошуку. Введемо наступні позначення.

A – масив, що містить вхідну відсортовану послідовність даних;

key – шуканий ключ;

low – індекс (номер) нижньої границі вхідної послідовності;

high **–** індекс (номер) верхньої границі вхідної послідовності;

mid – індекс (номер) середини структури даних;

Обчислення значення ***mid*** проводиться за такою формулою (5.1):

$$mid = low + (high - low)/2 \qquad (5.1)$$

При цьому повинні виконуватися наступні умови:

- якщо ***key < A[mid]***, то верхня межа ***high*** зміщується вліво (⬅) до початку масиву і обчислюється за формулою: ***high = mid-1***;
- якщо ***key > A[mid]***, то нижня межа ***low*** зміщується вправо (➡) до кінця масиву і обчислюється за формулою: ***low = mid + 1***.

Іншими словами, якщо шуканий ключ ***key*** менший за середнє значення ***A[mid]***, то подальший розгляд переноситься в ліву частину масиву, де знаходяться менші значення, а якщо значення ключа більше за середнє значення ***A[mid]***, розгляд переноситься в область більших значень (рис. 5.2).

	$mid = low + (high - low)/2$	
key < A[mid]		*key > A[mid]*
$high = mid - 1$		$low = mid + 1$

Рис. 5.2. Співвідношення для обчислення зміщення границь під час пошуку

Розглянемо задачу, яка полягає в необхідності визначити місце розташування (індекс в наведеному нижче масиві) ключа 33.

9	15	17	24	25	33	37	43	46	49
0	1	2	3	4	5	6	7	8	9

Для даної задачі значення границь дорівнюватимуть: ***low = 0***; ***high = 9***.

Середину масиву знайдемо за формулою:

mid = low + (high – low) / 2.

mid = 0 + (9-0) / 2 = 4.5 *(значення 4.5 округляється до мінімального цілого).*

mid = 4.

Таким чином визначена середина масиву. Значення, що відповідає середині масиву, дорівнює:

A[mid] = A[4] = 25; 25 <33.

Це означає, що нижню границю ***low*** необхідно пересунути вгору.

low = mid +1 = 4 + 1 = 5.

Далі розглядається права частина масиву.

9	15	17	24	25	33	37	43	46	49
0	1	2	3	4	5	6	7	8	9

Знову обчислюємо чергове значення середини масиву ***mid***.

mid = low + (high – low) / 2.

mid = 5 + (9 – 5) / 2 = 7.

Тепер значення середини нової ділянки дорівнює 7.

9	15	17	24	25	33	37	43	46	49
0	1	2	3	4	5	6	7	8	9

Порівнюємо значення, що зберігається в елементі масиву з індексом 7, з шуканим ключем 33.

Значення, збережене в місці розташування 7 дорівнює 43 і воно є більшим від того, яке ми шукаємо. Таким чином, шукане значення має перебувати в зліва від цього місця розташування.

9	15	17	24	25	33	37	43	46	49
0	1	2	3	4	5	6	7	8	9

high = mid – 1; high = 7 – 1 = 6; low = 5.

mid = low + (high – low) / 2.

mid = 5 – (6 – 5) / 2 = 5 + 0 = 5

Тепер обчислюємо середину знову. На цей раз отримуємо 5.

Це і є положенням шуканого ключа 33. Дійсно:

A[mid] = A[5] = 33.

Представимо даний процес роботи алгоритму двійкового пошуку в табличному вигляді (табл. 5.1).

Таблиця 5.1

Представлення роботи алгоритму в табличному вигляді

Імена параметрів	**Значення параметрів**		
	1-й крок	**2-й крок**	**3-й крок**
low	**0**	**low = mid + 1 = 4 + 1 = 5**	**5**
high	**9**	**9**	**high = mid – 1 = 7 – 1 = 6**
mid	**4**	**7**	**5**
A[mid]	**25**	**43**	**33**
	33 > 25	**33 < 43**	**33 = 33**
key = 33	**33**	**33**	**33**

5.2. Інтерполяційний метод пошуку

Інтерполяційний (інтерполюючий) пошук базується на принципі пошуку в телефонній книзі або, наприклад, в словнику. Замість порівняння кожного елемента з шуканим, як при лінійному пошуку, даний алгоритм здійснює прогноз місцезнаходження елемента: пошук відбувається подібно до бінарного пошуку, але замість поділу області пошуку на дві частини, інтерполяційний пошук проводить оцінку нової області пошуку за відстанню між ключем і поточним значенням елемента.

Іншими словами, бінарний пошук враховує лише знак різниці між ключем і поточним значенням, а інтерполяційний враховує також модуль цієї різниці і за даним значенням здійснює прогноз позиції наступного елемента для перевірки.

В середньому інтерполяційний пошук здійснює ***O(log(log(N)))*** операцій, де ***N*** – число елементів, серед яких проводиться пошук. Число необхідних операцій залежить від рівномірності розподілу значень серед елементів.

На практиці інтерполяційний пошук часто є більш швидким за бінарний, оскільки з обчислювальної сторони їх відрізняють лише арифметичні операції: інтерполяція – в інтерполяційному пошуку та розподіл на два – в бінарному, а швидкість їх обчислення відрізняється незначно, з іншого боку інтерполяційний пошук використовує таку важливу властивість даних, як однорідність розподілу значень. Ключем може бути не тільки номер, число, а й, наприклад, текстовий

рядок. Тоді стає зрозуміла аналогія з телефонною книгою: якщо ми шукаємо ім'я в телефонній книзі, що починається на «А», то потрібно шукати його на початку, але ніяк не у середині. В принципі, ключем може бути все що завгодно, тому що ті ж рядки, наприклад, запросто кодуються посимвольно. У найпростішому випадку символ можна закодувати значенням від 1 до 33 (тільки українські символи) або, наприклад, від 1 до 26 (тільки латинський алфавіт) і т.д.

Формула, що визначає алгоритм інтерполяційного пошуку, виглядає наступним чином:

$$mid = low + \frac{(key - A[low]) * (high - low)}{A[high] - A[low]}$$

Тут ***mid*** – номер елемента, з яким порівнюється значення ключа, ***key*** – ключ (шуканий елемент), ***A*** – масив упорядкованих елементів, ***low*** і ***high*** – номери крайніх елементів області пошуку. Важливо відзначити, що операція ділення в формулі є строго цілочисельною, тобто дробова частина, якою б вона не була, відкидається.

Якщо крайні елементи позначити іменами ***left*** і ***right***, то формула буде виглядати так:

$$mid = left + \frac{(key - A[left]) * (right - left)}{A[right] - A[left]}.$$

Завдання до Лабораторної роботи №5 для одновимірних масивів

1. Заповнити масив цілочисельними випадковими числами в діапазоні (10-100).
2. Розмір масиву ***N*** обчислюється як ціла частина від значення виразу
N = 20 + 0,6 * ***K***, де ***K*** – номер студента в журналі.
3. Відсортувати масив за методом бульбашки.
4. Знайти ключ, що знаходиться в середині лівої частини масиву.
5. Студентам з парними номерами варіантів використовувати метод бінарного пошуку, а з непарними – метод інтерполяційного пошуку.

ЛАБОРАТОРНА РОБОТА № 6

Лінійні однозв'язані і двозв'язані списки

Мета роботи: отримати навички в організації і обробці однозв'язаних і двозв'язаних списків, а також навчитися їх використовувати при вирішенні завдань.

Короткі теоретичні відомості

Зв'язний список – базова структура даних, в якій кожен елемент містить інформацію, необхідну для отримання наступного елемента. Основна перевага зв'язаних списків перед масивами полягає в можливості ефективного змінювати їх вигляд. За цю гнучкість доводиться жертвувати швидкістю доступу до довільного елементу списку, оскільки єдиний спосіб отримання елемента полягає у відстеженні зв'язків від початку списку.

Зв'язний список – це набір елементів, причому кожний з них є частиною вузла (*node*), який також містить посилання (*link*) на вузол. Вузли визначаються посиланнями на вузли, тому зв'язані списки іноді називають самопосилальними (*self-referent*) структурами. Більш того, хоча вузол зазвичай посилається на інший вузол, можливе посилання на самого себе, тому зв'язані списки можуть являти собою циклічні (*cyclic*) структури (рис. 6.1).

Рис. 6.1. Приклад взаємозв'язку елементів списків

Щоб задати динамічний список треба описати його вузол. Оскільки вузол складається з полів різних типів, то описати його можна неоднорідним типом – структурою.

Завдання типу елемента списку, який зберігає дані цілого типу

```
typedef struct Node {
    int item;
    Node * next;
} Node;
```

Щоб працювати зі списком як з єдиним об'єктом, треба ввести статичну змінну-покажчик, значення якої – це адреса першого (або заголовного) елемента списку. Якщо список порожній, вона повинна мати значення *NULL*.

Основними операціями обробки списку є:

1) пошук заданого елемента за його значенням або порядковим номером; операція завершується, коли елемент знайдений або переглянутий весь список (якщо елемента немає); результат операції повинен визначати, чи є елемент в списку чи ні і, якщо є, то можливо повернути його адресу або значення;

2) включення (вставка) в список нового елемента перед або після заданого елемента (в тому числі перед першим елементом або після останнього). Включенню, як правило, передує пошук елемента, після і/або перед яким відбувається включення; при включенні елемента перед першим в список змінюється заголовок списку; при включенні після деякого елемента змінюється поле посилання, після якого відбувається включення, тому треба визначати посилання на елемент, після якого відбувається включення (рис. 6.2);

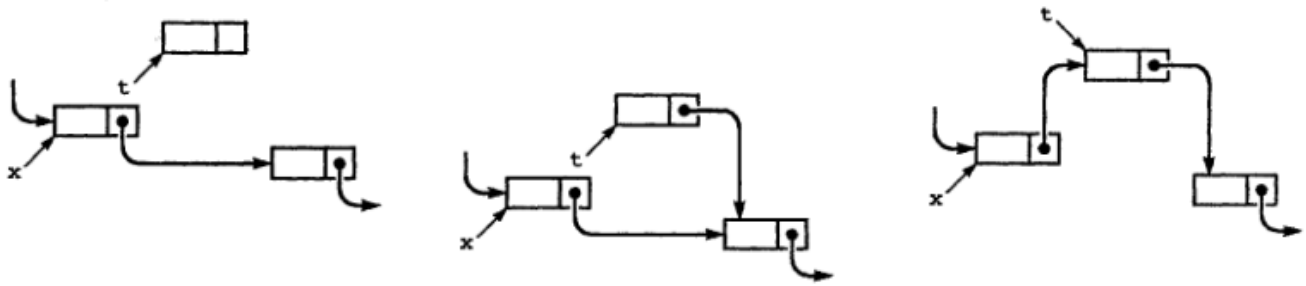

Рис. 6.2. Включення елементів списку

3) виключення (видалення) заданого елемента зі списку (в тому числі видалення першого елемента або останнього). Виключенню, як правило, передує пошук елемента, що виключається; результатом пошуку повинне бути посилання на елемент, що передує елементу, який виключається зі списку, так як при видаленні елемента зі списку змінюється поле посилання для елемента, що передує видаленому; при видаленні першого елемента змінюється заголовок списку (рис. 6.3);

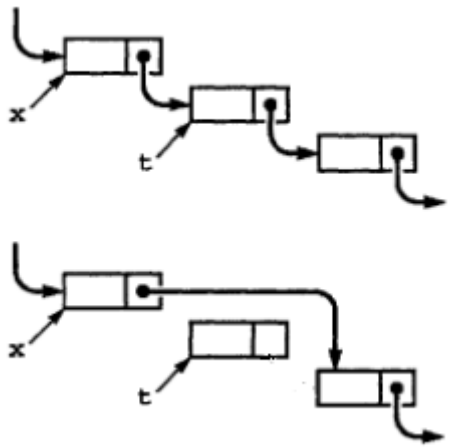

Рис. 6.3. Виключення заданого елемента зі списку

4) визначення числа елементів списку;

5) впорядкування елементів списку за значенням інформаційного поля.

Перевага зв'язаних списків перед масивами полягає в тому, що зв'язані списки ефективно змінюють розміри протягом життя. Зокрема, необов'язково заздалегідь знати максимальний розмір списку. Одне з важливих практичних наслідків цієї властивості полягає в можливості мати кілька структур даних, що мають спільний простір, не приділяючи особливої уваги їх відносному розміру в будь-який момент часу.

Реалізація простих операцій вставки і видалення елемента в списку:

```
// додавання нового елемента в початок списку
void push(Node **head, int data) {
    Node *tmp = (Node*) malloc(sizeof(Node));
    tmp-> item = data;
    tmp-> next = (*head);
    (*head) = tmp;

// видалення першого елемента в списку
int pop(Node **head) {
    Node* prev = NULL;
    int val;
    if (head == NULL) {
        exit(-1);
    }
    prev = (*head);
    val = prev-> item;
    (*head) = (*head)->next;
    free(prev);
    return val;
}
```

Реалізація функції друку елементів списку

```
void printLinkedList(const Node *head) {
    for (Node *t = head; t!= NULL; t = t->next) {
        printf("%d ", t->item);
    }
    printf("\n");
}
```

Динамічний список, в якому кожен елемент (крім, можливо, першого і останнього) пов'язаний з попереднім і наступним елементами, називається **двозв'язаним**. Кожен елемент такого списку має два поля з посиланнями: одне поле містить посилання на наступний елемент, інше поле – посилання на попередній елемент і третє поле – інформаційне. Наявність посилань на наступний і попередній вузли дозволяє рухатися по списку від кожного вузла в будь-якому напрямку: від вузла до кінця списку або від вузла до початку списку, поэтому такой список называют еще и двунаправленным (рис. 6.4).

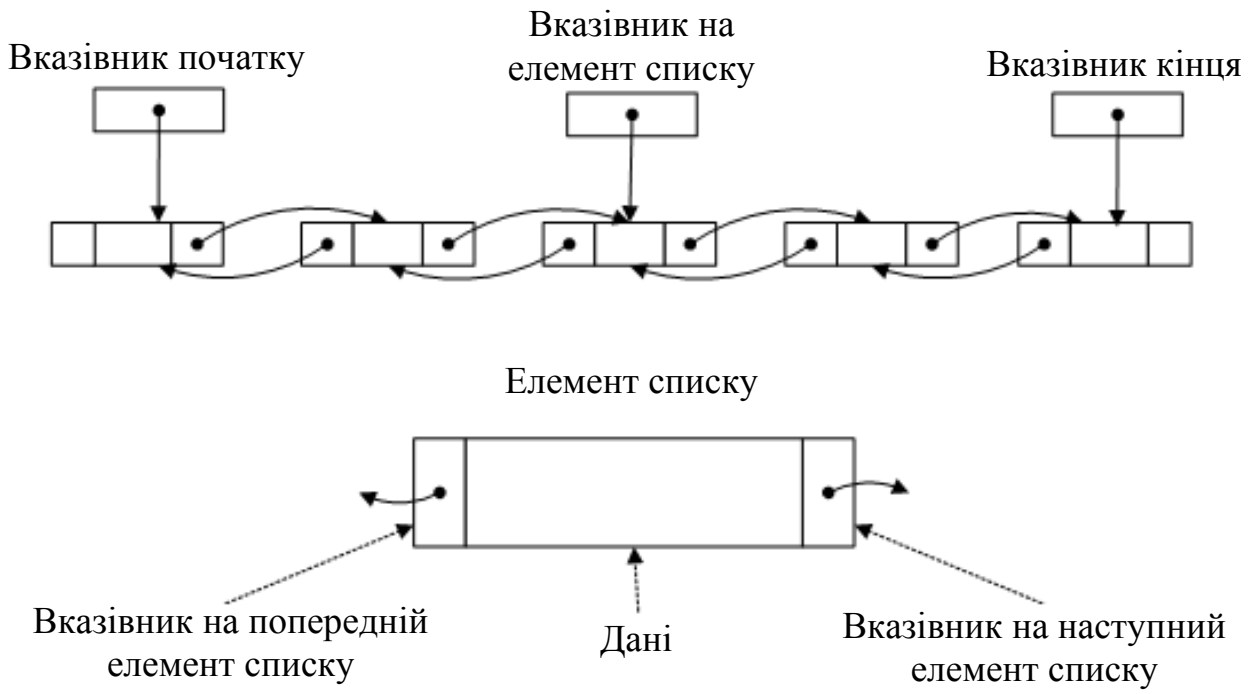

Рис. 6.4. Взаємодія елементів двонапрямлених списків

Завдання типу елемента двозв'язаного списку, який зберігає дані цілого типу:

```
typedef struct NodeT {
int item;
Node * next, * prev;
} NodeT;
```

Основні операції, що виконуються над двозв'язаним списком, ті ж, що і для однозв'язного списку. Оскільки двозв'язаний список більш гнучкий, ніж однозв'язний, то при включенні елемента в список, можна використовувати вказівник як на елемент, за яким відбувається включення, так і вказівник на елемент перед яким відбувається включення. При виключенні елемента зі списку можна використовувати як вказівник на сам елемент, що виключається, так і вказівник на елемент, що йому передує або наступний за ним. Але оскільки елемент двозв'язаного списку має два вказівника, то при виконанні операцій включення / виключення елемента треба змінювати більше зв'язків, ніж для однозв'язного списку. Пошук елемента двозв'язаного списку можна здійснювати:

а) переглядаючи елементи від початку до кінця списку,

б) переглядаючи елементи від кінця списку до початку,

в) переглядаючи список в обох напрямках одночасно: від початку до середини списку і від кінця до середини (враховуючи, що елементів в списку може бути парне або непарна кількість).

<u>Завдання до лабораторної роботи № 6.</u>

Для **кожного варіанта** – реалізувати однозв'язний і двозв'язний список і операції роботи з ними. Непарні варіанти зберігають в списку цілі значення, парні – символи. Списки заповнити з клавіатури. Операції:

- додати на початок;
- додати в кінець;
- додати в середину (після зазначеного за значенням елемента);
- видалити (з будь-якого місця списку);
- знайти за значенням;
- роздрукувати список.

Використовуючи два створені списки, виконати два завдання за варіантом. Варіант визначається номером студента у списку групи. Варіанти завдань визначати згідно з таблицею:

Варіант	**Завдання**	**Варіант**	**Завдання**	**Варіант**	**Завдання**
1	1, 13	**11**	1, 11	**21**	7, 13
2	2, 14	**12**	2, 12	**22**	8, 14
3	3, 11	**13**	3, 13	**23**	3, 5
4	4, 12	**14**	4, 14	**24**	6, 10
5	5, 9	**15**	1, 7	**25**	3, 7
6	6, 10	**16**	2, 6	**26**	4, 6
7	5, 7	**17**	3, 9	**27**	9, 13
8	6, 8	**18**	4, 8	**28**	10, 12
9	7, 9	**19**	5, 11		
10	8, 10	**20**	6, 12		

Завдання:

1. Створити новий (третій) однозв'язний список. Помістити в нього всі парні елементи з перших двох. Результуючий список вивести на екран.
2. Створити новий (третій) однозв'язний список. Помістити в нього перші три елементи з першого і другого списків. Результуючий список вивести на екран.
3. Створити новий (третій) однозв'язний список. Помістити в нього всі непарні елементи з перших двох. Результуючий список вивести на екран.
4. Створити новий (третій) однозв'язний список. Помістити в нього перші п'ять елементів з першого і другого списків. Результуючий список вивести на екран.
5. Створити новий (третій) однозв'язний список. Помістити в нього всі елементи з перших двох, які більше заданого числа. Результуючий список вивести на екран.
6. Створити новий (третій) однозв'язний список. Помістити в нього всі елементи з перших двох, які більше заданого символу. Результуючий список вивести на екран.

7. Створити новий (третій) однозв'язний список. Помістити в нього всі елементи з перших двох, які менше заданого числа. Результуючий список вивести на екран.

8. Створити новий (третій) однозв'язний список. Помістити в нього всі елементи з перших двох, які менше заданого символу. Результуючий список вивести на екран.

9. Створити новий (третій) однозв'язний список. Помістити в нього всі елементи з перших двох, за винятком нульового елемента. Результуючий список вивести на екран.

10. Створити новий (третій) однозв'язний список. Помістити в нього всі елементи з перших двох, за винятком символу *'a'*. Результуючий список вивести на екран.

11. Створити новий (третій) однозв'язний список. Помістити в нього всі додатні елементи з перших двох. Результуючий список вивести на екран.

12. Створити новий (третій) однозв'язний список. Помістити в нього всі символи нижнього регістра з перших двох. Результуючий список вивести на екран.

13. Створити новий (третій) однозв'язний список. Помістити в нього всі від’ємні елементи з перших двох. Результуючий список вивести на екран.

14. Створити новий (третій) однозв'язний список. Помістити в нього всі символи верхнього регістру з перших двох. Результуючий список вивести на екран.

ЛАБОРАТОРНА РОБОТА № 7

Структури даних: стек, черга, дерева

Мета роботи: сформувати практичні навички організації таких поширених структур даних як стеки, черги і дерева, а також їх використання при вирішенні завдань.

Теоретичні відомості

Стек – це список, у якого доступний один елемент (одна позиція). Цей елемент називається вершиною стека. Взяти елемент можна тільки з вершини стека, додати елемент можна тільки в вершину стека.

Черга – це список, у якого доступні два елементи (дві позиції) – початок і кінець черги. Помістити елемент можна тільки в кінець черги, а взяти елемент тільки з її початку.

Стеки і черги можуть бути організовані різними способами. При цьому вони можуть бути розміщені як в статичній пам'яті, так і в динамічній.

Приклад програми для роботи зі стеком, в якому реалізовані дві функції додавання елемента в вершину стека і вилучення елемента з подальшим його видаленням зі стека.

```
#include <stdio.h>
#include <stdlib.h>
// структура даних для зберігання інформації про один
// елемент стека
typedef struct Element {
   int value;
   struct Element* next;
}  Element;

void push(Element** first, int data);
int pop(Element** first, int* x);
//==================================
int main()
{
         Element* head = NULL; // вершина стека

         int i, x;
         for (i = 0; i<10; i++)
             push(&head, i);
         while (pop(&head, &x))
             printf("x = %d\n", x);
         return 0;
}
//===================================
void push(Element** first, int data)
{
       Element* temp;
```

```
        temp = (Element*)malloc(sizeof(Element));
        temp->value = data;
        temp->next = (*first);
        (*first) = temp;
}
//=====================================
int pop (Element** first, int* x)
{
        Element* temp;
        if (*first == NULL)
            return 0;
        temp = (*first);
        *x = temp->value;
        (*first) = temp->next;
        free (temp);

        return 1
}
```

Приклад роботи з чергою, в якому реалізовані дві функції додавання елемента в кінець черги і вилучення елемента з початку черги з подальшим його видаленням.

```
// =========== Приклад роботи з чергою ==
#include <stdio.h>
#include <stdlib.h>
// структура даних для зберігання інформації про один
// елементі черги
typedef struct Element {
   int value;
   struct Element* next;
}  Element;

void addElement(Element** first, Element** last, int data);
int extractElement(Element** first, Element** last, int* x);

int main()
{

      Element* head = NULL; // покажчик на перший елемент
      Element* tail = NULL; // покажчик на останній елемент

      int i, x;
      for (i = 0; i<10; i++)
          addElement(&head, &tail, i);
      while (extractElement(&head, &tail, &x))
          printf("x = %d\n", &x);
      return 0;
}
void addElement(Element** first, Element** last, int data)
```

```
{
      Element* temp;
      temp = (Element*)malloc(sizeof(Element));
      temp->value = data;
      temp->next = NULL;
      if (*last == NULL)
              (*first) = temp;
      else
              (*last)->next = temp;
      *last = temp;
}

int extractElement(Element** first, Element** last, int* x)
{
      Element* temp;
      if (*first == NULL)
          return 0;
      temp = (*first);
      *x = temp->value;
      (*first) = (*first)->next;
      if (*first == NULL)
          *last = NULL;
      free (temp);

      return 1;
}
```

Дерево – це кінцева множина T (можливо, порожня), що складається з одного або більше елементів (вузлів або вершин дерева) таких, що:

1) є один спеціально позначений елемент, що називається коренем даного дерева;

2) інші елементи містяться в $m > 0$ попарно непересічних множинах $T_1,...,T_m$, кожна з яких в свою чергу є деревом; дерева $T_1,...,T_m$ називаються піддеревами даного кореня (рис. 7.1а).

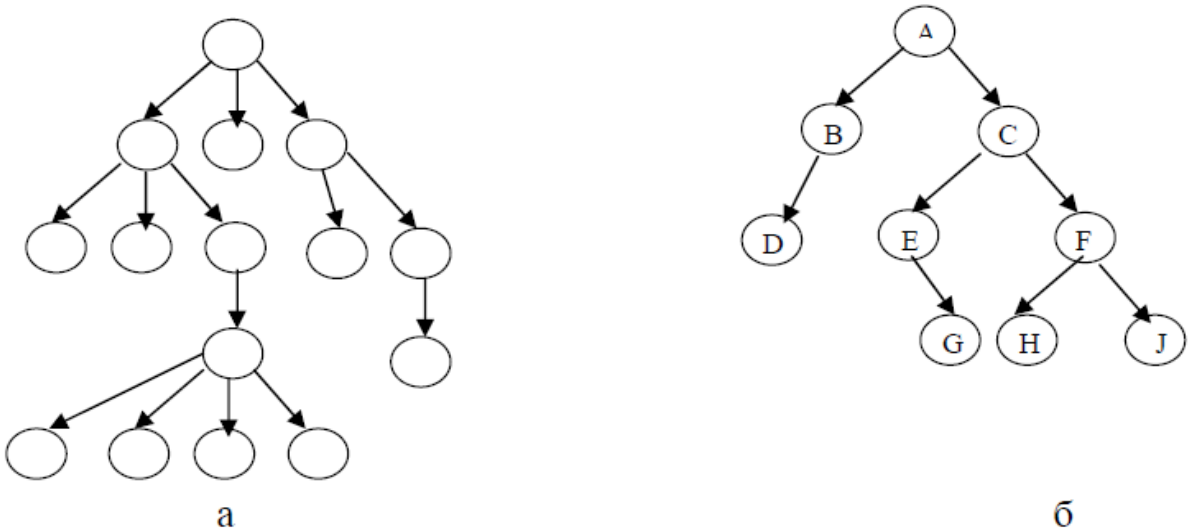

Рис. 7.1. Види дерев

Якщо має значення відносний порядок піддерев $T_1,...,T_m$, то кажуть, що дерево є впорядкованим. Число піддерев даного вузла називається ступенем цього вузла. Вузол з нульовим ступенем називається кінцевим вузлом (або листом, або термінальним вузлом), всі інші елементи – внутрішні вузли (нетермінальні). Максимальний ступінь всіх вершина називається ступенем дерева. Корінь дерева має рівень 0. Решта вершини мають рівень на одиницю більше рівня безпосереднього предка.

Максимальний рівень будь-якої з вершин називається **глибиною** або **висотою** дерева. Мінімальна висота при заданому числі вершин досягається, якщо на всіх рівнях, крім останнього, поміщається максимально можливе число вершин. Максимальне число вершин в дереві заввишки h досягається в тому випадку, якщо з кожної вершини, за винятком рівня h, виходять d піддерев, де d – ступінь дерева: на 0-му рівні 1 вершина, на 1-му – d_1 нащадків, на 2-му – d_2 нащадків, на 3-му рівні d_3 нащадків і т.д.

Найбільш широко використовуються виконавчі (бінарні) дерева (рис. 7.1б).

Бінарне дерево – це кінцева множина елементів, яке або є порожньою, або складається з кореня і з двох бінарних дерев, що не перетинаються і називаються лівим і правим піддеревами даного кореня. Таким чином, кожен елемент бінарного дерева має 0, 1 або 2 піддерев.

Бінарне дерево – впорядковане дерево, оскільки в ньому розрізняють ліве і праве піддерева.

Приклад програми для роботи з деревами

Нехай дано n-елементний масив цілих чисел. Необхідно сформувати ідеально збалансоване дерево з елементів масиву a. Потім видалити дерево.

Функція *createNode* (рис. 4.6) на вході отримує n – кількість елементів дерева, створює кореневу вершину і будує рекурсивно тим же способом ліве піддерево з $cnl = n / 2$ вершинами і праве піддерево з $cnr = n – nl – 1$ вершинами.

Функція *distruct* рекурсивно видаляє дерево.

```
// === Приклад роботи з деревами ========
#include <stdio.h>
#include <stdlib.h>
// структура даних для зберігання інформації про
// елементах дерева
typedef struct ELEMENT
{
      int data;
      struct ELEMENT *left, *right;
}  ELEMENT;

ELEMENT *createNode(long n, int *a, long *pi);
void distruct(ELEMENT *q);
void print(ELEMENT *q, long n);
```

```
int main()
{
      ELEMENT *root = NULL;
      long i;
      int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
      i = 0;
      root = createNode(10, a, &i);
      print(root, 0);
      distruct(root);
}
ELEMENT *createNode(long n, int *a, long *pi)
{
if (n <= 0)
    return NULL;
else
  {
    long nl, nr;
    ELEMENT *root;
    n1 = n/2; / * Число вершин в лівому поддереве * /
    nr = n – nl – 1; / * Число вершин у правому поддереве * /
    root = (ELEMENT *)malloc(sizeof(ELEMENT));
    root->data = a[*pi];
    (*pi)++;
    root->left = createNode(nl, a, pi);
    root->right = createNode(nr, a, pi);
    return root;
    }
}

void distruct(ELEMENT *q)
{
    if (q != NULL)
    {
        distruct(q->left);
        distruct(q->right);
        free(q);
      }
 }

void print(ELEMENT *q, long n)
{
      if (q != NULL)
      {
          long i;
          print(q->right,n+5);
          for (i = 0; i < n; i++)
```

```
                printf(“  “);
            printf(“%d\n”, q->data);
            print(q->left,n+5);
        }
}
```

<u>Завдання до лабораторної роботи № 7.</u>

Розробити програми для вирішення наведених нижче завдань.

<u>Примітка</u>: Студенти з парними номерами в журналі виконують завдання 1, 3, 5; з непарними – 2, 4, 6).

1. Заданий рядок тексту, який в тому числі містить множину дужок: *"("*, *"{"*, *"["* що відкриваються і закриваються. Рядок вважається коректним в тому випадку, якщо всім дужкам, що відкриваються, є відповідні дужки, що закриваються. Наприклад, рядок *"abc (as) [] {aa [z]}"* – вважається коректним, а рядок *"[[09] ({])"* – не є таким. Перевірити коректність рядка з використанням стека.
2. Розташувати елементи цілочисельного масиву розміром *N* в зворотному порядку з використанням стека.
3. Дано дві непусті черги, які містять однакову кількість елементів. Об'єднати черги в одну зі збереженням упорядкованості елементів.
4. Нехай є файл дійсних чисел і деяке число *C*. Використовуючи чергу, надрукувати спочатку всі елементи, що менші за числа *C*, а потім всі інші елементи.
5. Написати функцію, яка дозволяє визначити число входжень елемента *x* в бінарне дерево.
6. Знайти максимальний елемент бінарного дерева і кількість повторень максимального елемента в даному дереві.

ДОДАТОК 1

ПРИКЛАД ТИТУЛЬНОГО АРКУША

==========

Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

ЗВІТ
про виконання лабораторних робіт
з дисципліни
«Алгоритми та структури даних»

Виконав:
студент(ка) гр. _____

_____
(П.І.Б.)

Прийняв:

викладач. каф. ІСТ

_____
(П.І.Б.)

Дніпро
2020

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Алгоритми та структури даних : навч. Посібник / .О. Коротєєва. – Львів : Видавництво Львівської політехніки, 2014. – 280 с.
2. Алгоритми та структури даних: Навчальний посібник / В.М.Ткачук. – Івано-Франківськ: Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016. – 286 с.
3. Коротєєва Т.О. Алгоритми та структури даних : навч. посібник / Т. О. Коротєева. – Львів: Видавництво Львівської політехніки, 2014. – 280 с.
4. Трофименко О.Г. С++. Алгоритмізація та програмування : підручник / О.Г. Трофименко, Ю.В. Прокоп, Н.І. Логінова, О.В. Задерейко. 2-ге вид. перероб. і доповн. Одеса : Фенікс, 2019. – 477 с.
5. Advanced Data Structure and Algorithms / Produced & Printed by EXCEL BOOKS PRIVATE LIMITED, Phagwara Punjab, India : Lovely Professional University, 2011. – 293 p.
6. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. Addison-Wesley, 1985. – 427 p.
7. Aziz, Adnan. Algorithms For Interviews / Adnan Aziz. First Edition. CreateSpace Independent Publishing Platform. 2010. – 222 p.
8. Baka, Benjamin. Python Data Structures and Algorithms / Benjamin Baka. Packt Publishing Ltd. 2017. – 662 p.
9. Bhargava, Aditya. Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People / Aditya Bhargava. Manning. 2016. – 256 p.
10. Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduction To Algorithms / Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. 2d Edition. The MIT Press. 2001. – 984 p.
12. Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduction To Algorithms / Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. 3d Edition. The MIT Press. 2009. – 1292 p.
13. Dale, Nell., Walker, Henry Mackay. Abstract Data Types: Specifications, Implementations, and Applications. – Jones & Bartlett Learning, 1996. – 624 p.
14. Goodrich, Michael T. Tamassia,Roberto, Goldwasser, Michael H. Data Structures and Algorithms in Python / Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser. – N.Y.: John Wiley & Sons, Inc., 2013. – 768 p.
15. Goodrich, Michael T., Tamassia, Roberto. Algorithm Design. Foundations, Analysis, and Internet Examples / Michael T. Goodrich and Roberto Tamassia. – N.Y.: John Wiley & Sons, Inc., 2014. – 816 p.
16. Google C++ Style Guide, available at https://google.github.io/styleguide/cppguide.html (accessed 18 June 2020).
17. Heineman, George T., Pollice, Gary., Selkow, Stanley. Algorithms in a Nutshell / George T. Heineman, Gary Pollice, Stanley Selkow. 1st edition. O'Reilly Media. 2008. – 364 p.
18. Horowitr Ellis, Sahni Sartaj. Fundamentals of Data Structures / Ellis Horowitz, Sartaj Sahni. Computer Science Press. 1983. – 564 p.
19. Kleinberg, Jon., Tardos, Eva. Algorithm Design / Jon Kleinberg and Eva Tardos. 1st ed. Pearson Education, Inc. 2005. – 864 p.

20.Knuth, Donald. The Art of Computer Programming, Fundamental Algorithms / Donald Knuth. Volumes 1-4A Boxed Set. Third Edition. Addison-Wesley. 2011. - 3168 p.

21. Knuth, Donald. The Art of Computer Programming: Volume 1: Fundamental Algorithms. Third Edition. Addison-Wesley. 2011. – 650 p.

22. Knuth, Donald. The Art of Computer Programming: Volume 2: Seminumerical Algorithms. Third Edition. Addison-Wesley. 2011. – 762 p.

23. Knuth, Donald. The Art of Computer Programming: Volume 3: Sorting and Searching. Second Edition Addison-Wesley, 2011. – 780pp.

24. Knuth, Donald. The Art of Computer Programming: Volume 4A: Combinatorial Algorithms, Part 1. First Edition. Addison-Wesle. 2011. – 883 p.

25. Knuth, Donald. The Art of Computer Programming: Volume 4, Fascicle 5: Mathematical Preliminaries Redux; Backtracking; Dancing Links. Addison-Wesley. 2019. – 382 p.

26. Knuth, Donald. The Art of Computer Programming: Volume 4, Fascicle 6: Satisfiability. Addison-Wesley. 2015. – 310 p.

27. List_of_data_structures, available at https://en.wikipedia.org/wiki/List_of_data_structures (accessed 18 June 2021).

28. Manber, Udi. Introduction to Algorithms: A Creative Approach / Udi Manber. 1st Edition. Addison-Wesley. 1989. – 478 p.

29. Mcdowell, Gayle Laakmann. Cracking the Coding Interview. 189 Programming Questions and Solutions / Gayle Laakmann Mcdowell. 6th Edition. CareerCup, LLC. 2015. – 696 p.

30. The Stanford University C++ Style Guide, available at https://hownot2code.com/2017/01/18/the-stanford-university-c-style-guide/ (accessed 18 June 202**1**).

31. Sanjoy Dasgupta, Christos Papadimitriou and Umesh Vazirani, Algorithms. McGraw-Hill Education. 2006. – 336 p.

32. Sanjoy Dasgupta; Christos H Papadimitriou; Umesh Virkumar Vazirani. Algorithms. McGraw-Hill Higher Education. 2008. – 320 p.

33. Sedgewick R. Algorithms in C++, Parts 1-4: Fundamentals, Data Structure, Sorting, Searching / Robert Sedgewick. 3rd Edition. Addison-Wesley Professional. 1998. – 738 p.

34. Sedgewick. R., Wayne K. Algorithms. Fourth Edition. Addison-Wesley. 2011. – 955 p.

35. Shaffer, Clifford A. Data Structures and Algorithm Analysis. Edition 3.2 (C++ Version). Copyright © 2009-2012 by Clifford A. Shaffer. – 596 p.

36. Shaffer, Clifford A. Data Structures and Algorithm Analysis. Edition 3.2 (Java Version). Copyright © 2009-2013 by Clifford A. Shaffer. – 582 p.

37. Skiena S.S. The Algorithm Design Manual, Second Edition. Springer-Verlag London Limited, 2008. – 730 p.

38. Stephens, Rod. Essential Algorithms: A Practical Approach to Computer Algorithms / Rod Stephens. 1st Edition. Wiley. 2013. – 624 p.

39. Wirth, N. Algorithms and Data Structures / N. Wirth. Prentice Hall. 1985. - 288 p.

ДОДАТКОВА ЛІТЕРАТУРА

1. Вступ до програмування мовою С++. Організація обчислень : навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставровський. К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с.

2. Войтенко В.В., Морозов А.В.. С/С++. Теорія та практика : Навчально-методичний посібник. Видання 2 – виправлене. Житомир: 2004. – 324 с.

3. Дудзяний І.М.. Програмування мовою С++. Частина 1 : Парадигма процедурного програмування :навчальний посібник. Львів: ЛНУ імені Івана Франка, 2013. – 468 с.

4. С++. Основи програмування. Теорія та практика : підручник / О.Г. Трофименко, Ю.В. Прокоп, І.Г. Швайко, Л.М. Буката та ін. ; за ред. О.Г.Трофименко. Одеса: Фенікс, 2010. – 544 с.

5. Грицюк, Юрій, Рак, Тарас. Програмування мовою С++: Навчальний посібник / Юрій Грицюк, Тарас Рак. Львів: Львівський державний університет безпеки життєдіяльності, 2011. – 292 с.

6. Татарчук Д. Д., Діденко Ю. В. Програмування мовами С та С++: навч. посіб. / Д.Д. Татарчук, Ю.В. Діденко. К.: 2012. – 112 с.

7. Das, Vinu V. Principles of Data Structures Using C and C++ / Vinu V Das. New Age International (P) Ltd. 2006. – 360 p.

8. Davis, Stephen R. C++ For Dummies / Stephen R. Davis. 4Th Edition. Wiley. 2000. – 456 p.

9. Deitel, Paul, Deitel, Harvey. C++ How to Program. Introducing the New C++ 14 Standard. 10th Edition. Pearson Education. 2017. – 1080 p.

10. Franca, Paulo. C++: No Experience Required / Paulo Franca. Sybex Inc. 1997. - 597 p.

11. Lippman, Stanley B., Lajoie, Josee, Moo, Barbara E. C++ Primer / Stanley B. Lippman, Josee Lajoie, Barbara E. Moo. 5th Edition. Addison-Wesley. 2012. – 976 p.

12. McConnell, Steve. Code Complete / Steve McConnell. 2nd ed. Published by Microsoft Press. 2004 / 2011. – 914 p.

13. Overland, Brian. C++ Without Fear: A Beginner's Guide That Makes You Feel Smart / Brian Overland. 3rd Edition. Pearson. 2015. – 624 p.

14. Prata, Stephen. C++ Primer Plus / Stephen Prata. Sixth Edition. Addison-Wesley Professional. 2011. – 1440 p.

15. Schildt, Herbert. C++ from the Ground Up / Herbert Schildt. 3rd Edition. McGraw Hill. 2003. – 624 p.

16. Sebesta, Robert W. Concepts of programming Languages / Robert W. Sebesta. Eleventh Edition. Pearson Education Limited. 2016. – 788 p.

17. Stroustrup, Bjarne. Programming: Principles and Practice Using C++ / Bjarne Stroustrup. 2nd edition. Addison-Wesley Professional. 2014. – 1312 p.

Навчальне видання

Коротенко Григорій Михайлович
Коротенко Леонід Михайлович
Сергєєва Катерина Леонідівна
Грищенко Олена Володимирівна
Харь Альона Тарасівна

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з дисципліни
«Алгоритми і структри даних».

Для студентів факультету інформаційних технологій, що навчаються на спеціальностях 121 «Інженерія програмного забезпечення», 122 «Комп'ютерні науки», 124 «Системний аналіз», 126 «Інформаційні системи та технології».

Видано в світ
у Національному технічному університеті
«Дніпровська політехніка».
Свідоцтво про внесення до Державного реєстру ДК № 1842
4960050, м. Дніпро, просп. Д. Яворницького, 19