Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інформатика

Програмування та алгоритмічні мови

Лабораторний практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Електронні мікро- і наносистеми та технології»
спеціальності 176 «Мікро- та наносистемна техніка»

Укладачі: Ю. В. Вунтесмері, О. В. Семеновська

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2025

УДК 004.4 (075.8)
В19

Укладачі: *Вунтесмері Юрій Володимирович*, канд. техн. наук, доц.
Семеновська Олена Володимирівна, канд. техн. наук, доц.

Рецензент *Татарчук, Д.Д.*, д-р техн. наук, доц.,
НТУУ «*КПІ ім. Ігоря Сікорського*»

Відповідальний редактор *Тимофєєв, В.І.*, д-р техн. наук, проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 7 від 08.05.2025 р.)
за поданням вченої ради факультету/навчально-наукового інституту
(протокол № 04/2025 від 28.04.2025 р.)

Вунтесмері Ю. В.

В19 Інформатика. Програмування та алгоритмічні мови. [Електронний ресурс]: лаб. практикум: навч. посіб. для здобувачів ступеня бакалавра за освіт. програмою «Електронні мікро- і наносистеми та технології» спец. 176 «Мікро- та наносистемна техніка» / КПІ ім. Ігоря Сікорського; уклад.: Ю. В. Вунтесмері, О. В. Семеновська ; – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2025. – 97 с.

У посібнику викладено основи алгоритмізації та програмування з орієнтацією на задачі обробки інженерних даних; посібник містить короткі теоретичні відомості, приклади реалізації та інструкції до виконання лабораторних робіт; особливу увагу приділено роботі з масивами, списками, матрицями, а також алгоритмам статистичної обробки, сортування, аналізу сигналів і структурованих даних; наведено варіанти індивідуальних завдань; запропоновано комплексну систему самоконтролю.

Навчальний посібник призначений для здобувачів ступеня бакалавра за освітньою програмою «Електронні мікро- і наносистеми та технології»; за спеціальністю 176 «Мікро- та наносистемна техніка»; буде також корисним студентам суміжних спеціальностей, викладачам та усім, хто опановує основи прикладного програмування у технічному контексті.

УДК 004.4 (075.8)

Реєстр. № НП 24/25-512. Обсяг 6,56 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

З М І С Т

ВСТУП 4
ОСНОВНІ ВІДОМОСТІ ПРО ІСТОРІЮ ТА СТАНДАРТИ МОВИ С 6
Лабораторна робота № 1 Обробка дискретизованих сигналів 23
Лабораторна робота № 2 Статистична обробка одновимірного масиву 29
Лабораторна робота № 3 Похибки обчислень з плаваючою комою 35
Лабораторна робота № 4 Операції з двовимірними масивами 43
Лабораторна робота № 5 Операції з лінійними списками 49
Лабораторна робота № 6 Алгоритми сортування 56
Лабораторна робота № 7 Розв'язання систем лінійних рівнянь методом Гауса 64
Лабораторна робота № 8 Маніпуляції зі складними структурами. Нормалізація *WAVE*-файл 71
Лабораторна робота № 9 Знаходження опуклої оболонки множини точок на площині. Алгоритм Джарвіса 76
Лабораторна робота № 10 Стековий парсер 81
РОЗРАХУНКОВА РОБОТА 85
ПЕРЕЛІК ПОСИЛАНЬ 94
СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ 95

ВСТУП

Сучасна інженерна освіта неможлива без ґрунтовної підготовки у сфері інформаційних технологій та програмування. Вміння ефективно працювати з алгоритмами, розуміти принципи обробки даних, володіти мовами програмування є необхідною умовою для фахівців, які працюють у галузі мікро- та наносистемної техніки. У сучасних умовах розвитку науки й техніки, стрімкої цифровізації усіх сфер людської діяльності, особливої актуальності набуває забезпечення якісної фахової підготовки майбутніх інженерів у галузі інформаційних технологій. Інформатика та програмування сьогодні не лише є окремими складовими інженерної освіти, а й виступають універсальним інструментом для вирішення широкого спектра практичних завдань, які постають перед спеціалістами у сфері мікро- та наносистемної техніки. Саме тому ґрунтовне оволодіння алгоритмічним мисленням, знанням структур даних, цифрової обробки інформації та засобів програмування стає фундаментальною умовою формування конкурентоспроможного інженера нового покоління.

Пропонований методичний посібник містить лабораторний практикум з дисципліни «Інформатика. Програмування та алгоритмічні мови», який розроблено відповідно до освітньої програми підготовки бакалаврів технічного профілю. Матеріал посібника спрямовано на формування практичних навичок з інформатики та програмування, що мають безпосереднє прикладне значення у професійній діяльності. Основна увага приділяється не лише засвоєнню теоретичних засад предмету, а й розвитку практичних компетентностей через реалізацію лабораторних робіт, адаптованих до актуальних потреб галузі мікро- та наносистемної техніки.

Тематика лабораторного практикуму охоплює широкий спектр питань, які мають як фундаментальне, так і прикладне значення. Зокрема, розглядаються основи обробки масивів і сигналів, робота з простими і складними структурами даних, методи сортування та пошуку, обчислення з похибками, цифрова обробка інформації, основи чисельних методів. Усі лабораторні роботи мають чітку логічну структуру: визначення мети, короткий

виклад необхідної теорії, алгоритм виконання, приклади реалізації та контрольні запитання для самоперевірки. Такий підхід дозволяє студентам системно засвоювати навчальний матеріал, розвивати навички аналізу, логічного мислення, уважності до деталей, а також вміння працювати з технічною документацією та формувати звітність.

Однією з характерних особливостей посібника є його прикладна спрямованість. Кожне завдання орієнтоване на практичне застосування отриманих знань та моделювання реальних інженерних ситуацій. Це забезпечує не лише засвоєння базових алгоритмів, але й формування здатності самостійно вирішувати нестандартні задачі, здійснювати ефективний аналіз та інтерпретацію результатів, працювати з великими обсягами даних. Крім того, у методичному посібнику акцентовано увагу на роботі з динамічною пам’яттю, файловими структурами, числовими масивами, а також на використанні методів статистичного аналізу та цифрової обробки сигналів — знання, що є вкрай необхідними для технічних спеціальностей.

Реалізація практичних завдань передбачається за допомогою мов програмування з використанням стандартних бібліотек, що уможливлює їх універсальне застосування у різних середовищах. Студенти ознайомлюються з базовими конструкціями мов програмування, логікою побудови алгоритмів, структурним та об’єктно-орієнтованим підходами, що дозволяє їм швидко адаптуватися до сучасних вимог ринку праці.

Методичний посібник є важливим елементом навчального процесу у підготовці майбутніх інженерів. Його структура та зміст відповідають принципам поетапності, доступності та професійної орієнтації. Практикум спрямований не лише на формування знань і навичок, але й на розвиток у студентів самостійності, відповідальності, готовності до прийняття рішень, що є необхідними рисами сучасного інженера. Посібник може бути використаний як у рамках аудиторної роботи, так і для самостійного опрацювання матеріалу, забезпечуючи отже гнучкість і адаптивність освітнього процесу до індивідуальних потреб кожного здобувача освіти.

ОСНОВНІ ВІДОМОСТІ ПРО ІСТОРІЮ ТА СТАНДАРТИ МОВИ С

Мова С була винайдена і реалізована Деннісом Рітчі (Dennis Ritchie) для комп'ютера DEC PDP-11 в операційній системі Unix. Ця мова була розроблена на основі «старішої» мови BCPL, створеної свого часу Мартіном Річардсом (Martin Richards). BCPL справив певний вплив на мову B, розроблену Кеном Томпсоном (Ken Thompson). Своєю чергою, розвиток мови B призвів до створення у 1970 році мови С.

Протягом багатьох років стандартом С була фактично версія, що поставляється разом з операційною системою Unix. Ця версія вперше була описана Брайаном Керніганом (Brian Kernighan) і Деннісом Рітчі у книзі The C Programming Language (Englewood Cliffs, N.J.: Prentice-Hall, 1978). Влітку 1983 року було утворено комітет зі створення для мови С стандарту ANSI (American National Standards Institute - Національний інститут стандартизації США). Треба зазначити, що процес стандартизації зайняв досить чималий термін - шість років.

Стандарт C89

Стандарт ANSI було остаточно ухвалено у грудні 1989 року і вперше опубліковано на початку 1990 року. Цей стандарт був також ухвалений організацією ISO (International Standards Organization - Міжнародна організація зі стандартизації), тому він називається стандартом ANSI/ISO мови С. У 1995 році було ухвалено 1-шу Поправку до стандарту С, згідно з якою, серед іншого, було додано кілька бібліотечних функцій. У 1989 році стандарт С разом із 1-ю Поправкою став базовим документом Стандарту C++, що визначає С як підмножину C++. Версію С, визначену стандартом 1989 року, зазвичай називають C89.

Стандарт С99

Протягом 90-х років увага програмістів була прикута головним чином до розвитку стандарту C++. Тим часом розробка С також тривала, призвівши в 1999 році до появи стандарту С, який прийнято називати С99. Загалом С99 зберіг усі основні риси С89, тобто, можна сказати, що мова С залишилася самою собою! Комісія стандартизації С99 приділила основну увагу двом напрямам: додаванню кількох чисельних бібліотек і розвитку нових вузькоспеціальних засобів, таких як масиви змінної довжини і модифікатор вказівника restrict. Завдяки цим нововведенням мова С знову опинилася на передньому краї розвитку мов програмування.

Стандарт С99 вніс до мови С численні розширення та нові можливості

- **Вказівники, визначені з кваліфікаторами типу restrict**

Цей кваліфікатор застосовується тільки до вказівників. Вказівник, визначений із кваліфікатором типу restrict, від самого початку є єдиним засобом, за допомогою якого можна одержати доступ до об'єкта, що вказується. Доступ до об'єкта за допомогою іншого вказівника можливий лише тоді, коли цей другий вказівник заснований на першому. Отже, доступ до об'єкта можливий тільки для виразів, складених на основі вказівника з кваліфікатором типу restrict. Такі вказівники здебільшого використовуються як параметри функцій або для вказівки пам'яті, розподіленої за допомогою malloc(). Кваліфікатор типу restrict семантики програми не змінює.

- **Ключове слово inline**

Під час розробки С99 було додано ключове слово inline, яке застосовується до функцій. Ставлячи inline на початку оголошення функції, ви пропонуєте компілятору оптимізувати виклики до цієї функції. Зазвичай це означає, що під час компіляції код цієї функції буде вставлятися на місці викликів. Однак ключове слово inline є всього лише запитом до компілятора і може бути

проігнороване. У C99 особливо зазначено, що використання inline «передбачає, що виклики функції мають бути максимально швидкими». Специфікатор inline також підтримується в мові C++, і синтаксис C99 для цього ключового слова сумісний з C++.

Наприклад описана нижче функція max після оптимізації компілятором буде підставлена у точки виклику безпосередньо

```
inline int max(int a, int b)
{
     return a > b ? a : b;
}
```

- **Нові вбудовані типи даних**

У стандарті C99 з'явилися нові для C вбудовані типи даних.

_Bool

Один із нових типів даних, що з'явилися в C99, - це _Bool, у якому можна зберігати значення 1 і 0 (істина (true) і брехня (false)). _Bool являє собою цілий тип даних. Як відомо багатьом читачам, у мові C++ визначається ключове слово bool, яке, незважаючи на схожість, все ж відрізняється від _Bool. Отже, у написанні цього типу C99 і C++ несумісні. Крім того, у C++ визначаються вбудовані логічні константи true і false, а в C99 цього не робиться. Однак у C99 є заголовок <stdbool.h>, у якому визначено імена макросів bool, true і false. Отже, можна легко створювати код, сумісний з C/C++.

Причина того, що як ключове слово вказується _Bool, а не bool, полягає в тому, що у багатьох уже наявних C-програмах визначено їхні власні варіанти bool. Визначаючи логічний тип як _Bool, C99 дає можливість не змінювати вже написаний код. Однак у нові програми найкраще вставляти <stdbool.h>, а потім використовувати ім'я макроса bool.

_Complex і _Imaginary

Стандарт C99 з'явився разом із новою для C підтримкою арифметичних операцій з комплексними числами; ця підтримка містить у собі ключові слова _Complex і _Imaginary, додаткові заголовки та кілька нових бібліотечних

функцій. Однак жодних реалізацій не потрібно, щоб реалізувати типи уявних чисел (imaginary types), а автономні додатки (які обходяться без операційної системи) не зобов'язані підтримувати комплексні типи. Арифметичні операції з комплексними числами з'явилися в C99 для спрощення програмування чисельних методів.

Визначено такі комплексні типи:

float_Complex

float_Imaginary

double_Complex

double_Imaginary

long double_Complex

long double_Imaginary

Причина того, що як ключові слова визначено _Complex і _Imaginary, а не complex та imaginary, полягає у тому, що у багатьох наявних С-програмах уже визначено їхні власні типи комплексних даних, які використовують імена complex та imaginary. Визначаючи ключові слова _Complex і _Imaginary, C99 дозволяє не змінювати вже написаний код.

Заголовок <complex.h> визначає (крім усього іншого) макроси complex і imaginary, які у результаті макропідстановки перетворюються в _Complex і _Imaginary. Отже, у нові програми найкраще вставляти <complex.h>, а потім використовувати макроси complex і imaginary.

Типи цілих даних long long

У стандарті C99 з'явилися нові для С типи даних long long int і unsigned long long long int. Діапазон значень типу даних long long int не вужчий, ніж інтервал від $-(2^{63}-1)$ до $(2^{63}-1)$. А діапазон значень типу даних unsigned long long int зобов'язаний містити інтервал від 0 до $2^{64}-1$. Типи long long дозволяють підтримувати 64-розрядні цілі значення за допомогою вбудованого типу.

- **Масиви змінної довжини**

У С89 розмірності масивів необхідно оголошувати за допомогою виразів із цілих констант, а проте розмір масиву фіксується під час компіляції. Через

певні обставини, у C99 це правило було змінено. У C99 можна оголосити масив, розмірності якого визначаються будь-якими допустимими цілими виразами, зокрема й такими, значення яких стають відомі тільки під час виконання. Такий масив називається масивом змінної довжини (variable-length array, VLA). Однак такими масивами можуть бути тільки локальні масиви (тобто ті, у яких область видимості - прототип або блок). Ось приклад масиву змінної довжини:

```
int matrix[diml][dim2];
/* двовимірний масив змінної довжини */
```

У даному випадку розмір matrix визначається значеннями, що передаються через змінні dim1 і dim2. Отже, у результаті кожного виклику може вийти масив matrix із найрізноманітнішими вимірами.

Важливо зрозуміти, що масиви змінної довжини за час «свого життя» не змінюють своїх розмірів. (Іншими словами, вони не є динамічними.) Насправді масив змінної довжини створюється з іншим розміром щоразу, коли зустрічається його оголошення.

Можна вказати масив змінної довжини незазначеного розміру, використовуючи як розмір зірочку, *.

Поява масивів змінної довжини спричинила невелику зміну в операторі sizeof. Взагалі кажучи, sizeof - це оператор, який обчислюється під час компіляції. Тобто під час компіляції він зазвичай перетворюється на цілу константу, значення якої дорівнює розміру типу або об'єкта. Однак якщо sizeof застосовується до масиву змінної довжини, то своє значення він отримує тільки під час виконання. Ця зміна була необхідна тому, що розмір масиву змінної довжини не можна дізнатися до часу виконання.

Однією з головних причин появи масивів змінної довжини є бажання спростити програмування чисельних методів. Звичайно, цей засіб застосовується досить широко. Але пам'ятайте - масиви змінної довжини не підтримуються Стандартом C89 (і в мові C++).

- **Використання кваліфікаторів типів в оголошенні масиву**

У С99 за оголошення масиву як параметра функції, всередині квадратних дужок цього оголошення можна вказати ключове слово static. Воно повідомляє компілятору, що у масиві, на який вказує цей параметр, завжди буде знаходитися як мінімум названа кількість елементів. Наприклад:

```
int f(char str[static 80])
{
 // тут str завжди є вказівником
 // на масив із 80 елементів
 //
}
```

Тут надається гарантія, що str вказуватиме на початок масиву типу char, а проте у ньому буде не менше 80 елементів.

Усередині квадратних дужок також допускаються ключові слова restrict, volatile і const, але тільки для параметрів функцій. Використання restrict означає, що вказівник є єдиним засобом доступу до об'єкта. Застосування const показує, що вказівник вказує на один і той самий масив (тобто вказівник завжди вказує на один і той самий об'єкт).

- **Однорядкові коментарі**

Завдяки Стандарту С99, у мові С з'явилися однорядкові коментарі. Коментар такого виду починається з // і доходить до кінця рядка. Наприклад,

```
// Це коментар
int i; // це інший коментар
```

- **Розділення коду та оголошень**

Відповідно до Стандарту С89 усі оголошення, що знаходяться всередині блоку, повинні передувати першому оператору коду. Стандарт С99 дозволяє оголошувати змінні за місцем ініціалізації.

- **Оголошення змінних усередині циклу for**

С99 розширює можливості циклу for, дозволяючи оголошення однієї або декількох змінних у частині ініціалізації циклу. Область видимості змінної, оголошеної таким способом, обмежена блоком програми, керованим виразом

for. Тобто змінна, оголошена всередині циклу for, буде локалізована всередині цього циклу. Ця можливість з'явилася у мові С тому, що керуюча змінна циклу for часто необхідна тільки всередині цього циклу. А оскільки ця змінна локалізована всередині циклу, то вдається уникнути непотрібних побічних ефектів.

- **Складені літерали**

С99 дає можливість визначати складені літерали, які є виразами, що складаються з масивів, структур або об'єднань; ці вирази і позначають об'єкти даного типу. Складений літерал створюється шляхом вказівки імені типу у круглих дужках, за яким слідує список ініціалізації, обов'язково укладений у фігурні дужки. Коли ім'ям типу є масив, то розмір вказувати не можна. Створюється безіменний об'єкт. Ось приклад складеного літерала:

```
double *fp = (double[]) {1.0, 2.0, 3.0};
```

У даному випадку створюється вказівник на double, що має назву fp і вказує на перший елемент масиву, що складається з трьох елементів типу double.

Складений літерал, створений в області видимості файлу, існує весь час життя програми. А складений літерал, створений усередині блоку, є локальним об'єктом, який зруйнується, щойно у разі виконання програми відбудеться вихід із цього блоку.

- **Масиви зі змінними межами як члени структур**

С99 дає можливість як останній член структури вказувати масив без розміру. (У структурі перед гнучким масивом-членом має стояти як мінімум ще один член.) Він називається членом-масивом зі змінними межами. Отже, структура може мати як член масив змінного розміру. У розмірі такої структури, що повертається sizeof, пам'ять для гнучкого масиву не враховується.

Зазвичай пам'ять для структури з членом-масивом зі змінними межами розподіляється автоматично, за допомогою malloc(). Крім розміру структури, необхідно ще виділити додаткову пам'ять, щоб розмістити член-масив зі

змінними межами потрібного розміру. Наприклад, якщо є таке визначення структури

```
sturct mystruct {
 int a;
 int b;
 float fa[]; //масив зі змінними границями
};
```

то у разі виконання наступного коду буде виділятися місце для масиву з 10 елементів:

```
struct mystruct *p;
p = (struct mystruct *) malloc(sizof(struct mystuct) + 10 *sizof(float);)
```

Оскільки sizeof (struct mystruct) дає значення, в якому не враховано розмір пам'яті для fa, то у разі виклику malloc() за допомогою виразу 10 *sizof(float) додатково виділяється місце для розміщення масиву з 10 елементів типу float.

- **Призначені ініціалізатори**

У С99 з'явилася нова для С можливість, яка буде особливо корисною для програмістів, які працюють з розрідженими масивами. Це призначені ініціалізатори. Такі ініціалізатори бувають двох видів: одного виду - для масивів, а іншого - для структур і об'єднань. Для масивів використовуються призначені ініціалізатори такого виду:

[індекс] = знач

де індекс вказує елемент, ініціалізований за допомогою значення знач (тобто той елемент, якому присвоюється початкове значення знач). Наприклад,

int a[10] = { [0] = 100, [3] = 200};

У даному випадку ініціалізуються тільки елементи з індексами 0 і 3.

Для членів структур або об'єднань використовуються призначені ініціалізатори такого вигляду:

.ім'я-члена

Застосування до структури призначеного ініціалізатора дозволяє легко ініціалізувати тільки потрібні члени структури. Наприклад,

```
struct mystruct {
 int a;
 int b;
 int c;
} ob = { .c = 30, .a = 10 };
```

У цьому випадку член b залишається неініціалізованим.

Крім того, застосування призначених ініціалізаторів дає можливість ініціалізувати структуру, навіть не знаючи порядку розташування її членів. Це корисно для зумовлених структур, таких як div_t, або для структур, визначених деякими незалежними виробниками.

- **Нові можливості сімейства функцій printf() і scanf()**

У С99 для сімейства функцій printf() і scanf() передбачено нову можливість: вони можуть маніпулювати з типами даних long long int і unsigned long long int. Модифікатором формату для long long є ll. Наприклад, у такому фрагменті показано, як виводити значення типу long long int і unsigned long long int:

```
long long int val;
unsigned long long int u_val;
printf("%lld %llu", val, u_val);
```

Модифікатор ll можна застосовувати до специфікаторів формату: d, i, o, u і x - як для printf(), так і для scanf().

У С99 додано модифікатор hh, який застосовується для вказівки char-аргумента разом зі специфікаторами формату: d, i, o, u і x.

Обидва модифікатори, ll і hh, можна використовувати також разом зі специфікатором n.

Специфікатори формату a і A, які були додані до printf(), змушують виводити значення з плаваючою комою у шістнадцятковому форматі. Формат значення виходить такий:

```
[-]0xh.hhhhhhp+d
```

Якщо використовується A, то x і p виводитимуться на верхньому регістрі. Специфікатори формату a і A були також додані до scanf() і вони читають значення з плаваючою комою.

- **Нові бібліотеки C99**

У C99 додані нові бібліотеки та заголовки:

<complex.h> Підтримує арифметичні операції із комплексними числами.

<fenv.h> Дає доступ до прапорців стану обчислювача, що виконує операції з плаваючою комою та іншими сигналами цього обчислювача.

<inttypes.h> Визначає стандартний набір імен цілих типів, що переноситься. Також підтримує функції, які працюють із цілими значеннями найбільшої розрядності.

<iso646.h> Додано у 1995 році Поправкою 1. Визначає імена макросів, які відповідають різним операторам, таким як && і ^.

<stdbool.h> Підтримує типи логічних даних. Визначає імена макросів bool, true та false, що допомагає забезпечувати сумісність із C++.

<stdint.h> Визначає стандартний набір імен цілих типів, що переноситься. Цей заголовок входить до складу <inttypes.h>.

<tgmath.h> Визначає макроси для родового (абстрактного) типу чисел із плаваючою точкою.

<wchar.h> Доданий у 1995 році Поправкою 1. Підтримує багатобайтові та двобайтові строкові функції.

<wctype.h> Додано у 1995 році Поправкою 1. Підтримує багатобайтні та двобайтові строкові функції класифікації.

- **Зарезервований ідентифікатор __func__**

У C99 визначено ідентифікатор __func__, який вказує (як строкового літерала) ім'я функції, у якій зустрічається __func__.

- **Розширені цілі типи**

C99 <stdint.h> визначає кілька розширених цілих типів. Розширені типи включають типи з точною розрядністю, мінімальною розрядністю, максимальною розрядністю і найшвидший цілий тип. Ось добірка таких типів:

int16_t	Тип 16-розрядних цілих
int_least16_t	Тип цілих, що містить не менше 16 розрядів
int_fast32_t	Найшвидший тип цілих, що містить не менше 32 розрядів
intmax_t	Тип найбільших цілих
uintmax_t	Тип найбільших цілих без знаку

- **Зміни у правилах просування цілих типів**

У C99 розширено правила просування цілих типів. У C89 значення типу char, short int або бітового поля int можна було використовувати у виразі замість int або unsigned int. Якщо просунуте значення містилося в int, просування виконувалося до int; в іншому випадку початкове значення просувалося до unsigned int.

У C99 кожному цілому типу присвоєно ранг. Наприклад, ранг long long int вищий, ніж ранг int, який у свою чергу вищий, ніж ранг char і так далі. У виразі будь-який цілий тип, ранг якого нижче, ніж ранг int або unsigned int, може використовуватися замість int або unsigned int.

Стандарт C11

У 2007 році почалася робота над іншою редакцією стандарту C, який неофіційно називався «C1X» до його офіційної публікації ISO/IEC 9899:2011 8 грудня 2011 року. Комітет стандартів C прийняв рекомендації щодо обмеження впровадження нових функцій, які не були перевірені існуючими реалізаціями.

Стандарт C11 додає численні нові функції до C і бібліотеки, включаючи загальні макроси типу, анонімні структури, покращену підтримку Unicode, атомарні операції, багатопотоковість і функції перевірки меж. Він також робить деякі частини існуючої бібліотеки C99 необов'язковими та покращує сумісність

із C++. Стандартний макрос __STDC_VERSION__ визначається як 201112L, щоб вказати, що підтримка C11 доступна.

Основні розширення стандарту C11 порівняно із C99

- Специфікація вирівнювання (специфікатор _Alignas, оператор _Alignof, функція aligned_alloc, заголовок <stdalign.h>)
- Специфікатор функції _Noreturn і заголовок <stdnoreturn.h>
- Типові загальні вирази за допомогою ключового слова _Generic. Наприклад, наступний макрос cbrt(x) перетворюється на cbrtl(x), cbrt(x) або cbrtf(x) залежно від типу x:

```
#define cbrt(x) _Generic((x), long double: cbrtl, \
        default: cbrt, \
         float: cbrtf)(x)
```

- Підтримка багатопотокової обробки (специфікатор класу зберігання _Thread_local, заголовок <threads.h>, включаючи функції створення/керування потоком, м'ютекс, змінну умови та функціональність зберігання, що залежить від потоку, а також <stdatomic.h> для атомарних операцій, що підтримують модель пам'яті C11).
- Покращена підтримка Unicode на основі технічного звіту C Unicode (типи char16_t і char32_t для зберігання даних у кодуванні UTF-16/UTF-32, включаючи функції перетворення в <uchar.h> і відповідні префікси рядкових літералів u і U, а також префікс u8 для літералів у кодуванні UTF-8).
- Видалення функції gets (на користь безпечніших fgets), яка була застарілою у попередній версії стандарту мови C, ISO/IEC 9899:1999/Cor.3:2007(E).

- Додаткові макроси для запиту характеристик типів із плаваючою комою, що стосуються субнормальних чисел із плаваючою комою та кількості десяткових цифр, які тип може зберігати.
- Анонімні структури та об’єднання, корисні, коли об’єднання та структури є вкладеними, наприклад
  struct T { int тег; об'єднання { float x; int n; }; };.
- Статичні твердження, які оцінюються під час трансляції на пізнішому етапі, ніж #if і #error, коли типи зрозумілі транслятору.
- Ексклюзивний режим створення та відкриття (суфікс «…x») для fopen. Це обробляється як O_CREAT|O_EXCL у POSIX, який зазвичай використовується для файлів блокування.
- Функція quick_exit як третій спосіб завершити програму, призначена для виконання принаймні мінімальної деініціалізації.
- Нова функція timespec_get і відповідна структура в <time.h> зі ступенем сумісності з POSIX.

Стандарт C17

C17 — це неофіційна назва для ISO/IEC 9899:2018, стандарту для мови програмування C, опублікованого у червні 2018 року. Він не містить нових функцій мови, лише технічні виправлення та роз’яснення щодо недоліків у C11. Стандартний макрос __STDC_VERSION__ визначається як 201710L, щоб вказати, що підтримка C17 доступна.

Стандарт C23

C23 — неофіційна назва поточної на момент укладання посібника основної версії стандарту мови C. Протягом більшої частини свого розвитку він був неофіційно відомий як "C2X". C23 був опублікований у жовтні 2024 року як

ISO/IEC 9899:2024. Стандартний макрос __STDC_VERSION__ визначається як 202311L, щоб вказати, що підтримка C23 доступна.

Зміни, інтегровані в останню робочу версію C23.

Стандартна бібліотека

- Додано функцію memset_explicit() у <string.h> для стирання чутливих даних, де збереження пам'яті повинно виконуватися завжди, незалежно від оптимізацій.
- Додано функцію memccpy() у <string.h> для ефективного об'єднання рядків - аналогічно до розширень POSIX і SVID C.
- Додано функції strdup() та strndup() у <string.h> для виділення копії рядка - подібно до розширень POSIX та SVID C.
- Додано функцію memalignment() у <stdlib.h> для визначення байтового вирівнювання вказівника.
- Додано бітові утиліти / макроси / типи у новому заголовку <stdbit.h> для дослідження багатьох цілих типів. Всі починаються з stdc_, щоб мінімізувати конфлікт зі старим кодом і сторонніми бібліотеками.
- Додано функції stdc_count_ones*() та stdc_count_zeros*() для підрахунку кількості 1 або 0 бітів у значенні.
- Додано функції stdc_leading_ones*() та stdc_leading_zeros*() для підрахунку початкових 1 або 0 бітів у значенні.
- Додано функції stdc_trailing_ones*() та stdc_trailing_zeros*() для підрахунку кінцевих 1 або 0 бітів у значенні.
- Додано функції stdc_first_leading_one*() та stdc_first_leading_zero*() для пошуку першого ведучого біта зі значенням 1 або 0.
- Додано функції stdc_first_trailing_one*() та stdc_first_trailing_zero*() для пошуку першого заднього біта зі значенням 1 або 0.
- Додано stdc_has_single_bit*() для визначення, чи value є точним степенем 2 (повертає true тоді і тільки тоді, коли є єдиний біт 1).

- Додано stdc_bit_floor*() для визначення найбільшого інтегрального степеня 2, який не перевищує value.
- Додано функцію stdc_bit_ceil*() для визначення найменшого інтегрального степеня від 2, не меншого за value.
- Додано функцію stdc_bit_width*() для визначення кількості бітів для представлення значення.
- Додано функцію timegm() у <time.h> для перетворення часової структури у значення календарного часу - подібно до функції у бібліотеках glibc та musl.
- Нові функції <math.h> на основі рекомендацій IEEE 754-2019, такі як функції тригонометрії, що працюють з частинами пі та exp10.
- Додано специфікатор двійкового перетворення %b до сімейства функцій printf().
- Додано специфікатор двійкового перетворення %b до сімейства функцій scanf().
- Додано підтримку двійкового перетворення 0b та 0B до сімейств функцій strtol() та wcstol().
- Зроблено так, щоб функції bsearch(), bsearch_s(), memchr(), strchr(), strpbrk(), strrchr(), strstr() та їх широкі аналоги wmemchr(), wcschr(), wcspbrk(), wcsrchr(), wcsstr() повертали об'єкт з кваліфікацією const, якщо їм було передано такий об'єкт.

Препроцесор

- Додано директиви #elifdef та #elifndef, які по суті еквівалентні #elif defined та #elif !defined. Обидві директиви було додано до стандарту C++23 та GCC 12.
- Додано директиву #embed для включення бінарних ресурсів та __has_embed, що дозволяє перевіряти наявність ресурсу директивами препроцесора.
- Додано директиву #warning для діагностики.

- Додано __has_include, що дозволяє перевіряти наявність заголовка директивами препроцесора.
- Додано __has_c_attribute, що дозволяє перевіряти наявність атрибута за допомогою директив препроцесора.
- Додано функціональний макрос __VA_OPT__ для варіаційних макросів, який розширюється до свого аргументу, тільки якщо варіаційний аргумент було передано до макросу, що його містить.

Типи

- Додано nullptr_t, тип нульового вказівника.
- Додано типи _BitInt(N) та unsigned _BitInt(N) для цілих чисел з точністю до біта. Додано макрос BITINT_MAXWIDTH для максимальної ширини бітів.
- Додано макроси ckd_add(), ckd_sub(), ckd_mul() для перевірених цілочисельних операцій.
- Змінно-модифіковані типи (але не VLA, які є автоматичними змінними, що виділяються у стеку) стають обов'язковою функцією.
- Покращено підтримку використання const з масивами.
- Стандартизовано оператор typeof(...).
- Значення ключового слова auto було змінено, щоб викликати виведення типів, зберігши його старе значення специфікатора класу сховища, якщо воно використовується разом з типом. На відміну від C++, C23 дозволяє виведення типів лише для визначень об'єктів (без виведення типу повернення функції або типу параметра функції).
- Змінено правила сумісності для структурних, об'єднань та перелічуваних типів, щоб дозволити перевизначення сумісного типу з тим самим тегом.
- Ціле число точної ширини тепер може перевищувати значення intmax_t (N2888)

Константи

- Додано константу nullptr для типу nullptr_t.
- Додано суфікси wb та uwb для цілих літералів типу _BitInt(N) та беззнакових типів _BitInt(N), наприклад, 6uwb дає беззнаковий _BitInt(3), а -6wb дає знаковий _BitInt(4), який має три біти значення та один біт знаку.
- Додано префікси 0b та 0B для двійкових літералів, наприклад, 0b1010101010 (що дорівнює 0xAA).
- Додано роздільник ' до буквених констант, наприклад, 0xFE'DC'BA'98 (що відповідає 0xFEDCBA98), 299'792'458 (що відповідає 299792458), 1.414'213'562 (що відповідає 1.414213562).
- Додано можливість вказувати базовий тип числення.
- Дозволено перелікам enum без фіксованого базового типу зберігати значення, які не можна представити за допомогою типу int.

Лабораторна робота № 1
Обробка дискретизованих сигналів

Мета роботи: Формування практичних навичок роботи з дискретизованими сигналами та їх обробки шляхом реалізації алгоритмів обчислення похідних, фільтрації, пошуку екстремумів та точок перетину.

Короткі теоретичні відомості

У сучасних інформаційно-комунікаційних технологіях обробка сигналів є фундаментальним процесом, що охоплює передачу, збереження та аналіз даних. У багатьох випадках початкові сигнали мають аналогову природу, тобто є безперервними у часі та значенні. Для того, щоб такі сигнали могли бути оброблені за допомогою цифрових пристроїв, необхідно здійснити їх перетворення у дискретну форму. Цей процес називається дискретизацією.

Аналоговий сигнал — це сигнал, який є неперервною функцією часу (або іншої незалежної змінної) і може набувати будь-яких значень з деякого діапазону. У протилежність йому, дискретний сигнал визначається лише у вибраних (дискретних) моментах часу та має скінченну або квантизовану кількість можливих значень.

Дискретизація — це процес перетворення неперервного сигналу у послідовність відліків, тобто значень сигналу у певні моменти часу. Цей процес є першим етапом аналого-цифрового перетворення (АЦП).

Дискретизований сигнал подається до обробки у вигляді одномірного масиву реальних або цілих чисел, кожне з яких має сенс амплітуди сигналу у певний момент часу. Сусідні елементи масиву представляють відліки сигналу у моменти, що відрізняються на інтервал дискретизації. Величина обернена до інтервалу дискретизації називається частотою дискретизації.

Отже робота зводиться до завантаження масиву дискретизованого сигналу з файлу до пам'яті і подальшій його обробці.

Вичитування одномірного масиву з файлу

Для читання масиву, розмір якого наперед невідомий, слід використовувати динамічне виділення пам'яті [1]. На початку задати масив довжиною 1 елемент та змінну, що має зберігати його поточний розмір.

```
int n = 1;
double *a;
a = (double*)malloc(sizeof(double));
```

Далі почати читати з файлу по одному елементу з даного типу, зберігаючи прочитане у останньому елементі масиву. Вважаємо, що змінна *fd* вже містить вказівник, отриманий за допомогою *fopen.*

```
while(!feof(fd)){
        fscanf(fd,"%e",&a[n-1]);
```

Після кожного прочитаного значення збільшувати розмір масиву на 1.

```
        a = realloc(a,++n*sizeof(double));
```

Після виходу з циклу (закінчення файлу) масив матиме один зайвий останній елемент. Його простіше проігнорувати, зменшивши лічильник розміру до реального значення.

```
}
n--;
```

Наразі маємо масив *a* розміром *n*, яким можна користуватись так само, як статичними масивами.

Усереднення масиву

Середнє значення $S = \frac{1}{n}\sum_{i=0}^{n-1} a_i$

Середньоквадратичне значення $SS = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} a_i^2}$

Похідні дискретизованого сигналу

Перша похідна (центральна різниця): $a_i = (a_{i+1} - a_{i-1})/2\Delta t$

Друга похідна (центральна різниця): $a_i = (a_{i+1} - a_{i-1})/\Delta t = (a_{i+2} - 2a_i + a_{i-2})/\Delta t^2$

Для першої та останньої точки похідні не обчислюються, а приймаються рівними значенням у сусідніх точках.

Рекурсивний фільтр

Цифровий рекурсивний фільтр використовується для обробки сигналів у реальному часі. Поточне значення відфільтрованої величини залежить не тільки від поточного та (M-1) попередніх значень вхідної величини, але також і від M попередніх значень вихідної величини.

Масив значень відфільтрованої величини: $y_i = \sum_{j=0}^{M1} b_j x_{i-j} - \sum_{k=1}^{M2} a_k y_{i-k}$,

де $i = M \ldots N-1$, $M = \max\{M1, M2\}$ – порядок фільтра, a,b – масиви коефіцієнтів.

Спрощений запис для фільтра другого порядку:

$$y_i = -a_1 y_{i-1} + a_2 y_{i-2} + b_0 x_i + b_1 x_{i-1} + b_2 x_{i-2}$$

Умови локальних екстремумів

Умова локального максимуму: $a_{i-1} < a_i > a_{i+1}$

Умова локального мінімуму: $a_{i-1} > a_i < a_{i+1}$

Пошук значення дискретизованої функції

Спочатку знаходять таке i, що $a_i < A < a_{i+1}$, або $a_i = A$, в останньому випадку результат $T = i\Delta t$ вже отриманий. Далі обчислюють відстань від найближчого вузла до точки перетину та координату точки перетину, вважаючи функцію на відрізку лінійною: $T = i\Delta t + \frac{A - a_i}{a_{i+1} - a_i}\Delta t$.

Програма роботи:

1. Обрати файл (*l1vXX.txt*, де *XX* – номер варіанту) з каталогу вхідних даних для лабораторних робіт відповідно до номеру варіанту (див. табл. 1). Прочитати вміст файлу у одномірний масив типу *double*, виділивши під нього пам'ять динамічно. Надалі вважати масив значеннями дискретизованого сигналу із періодом дискретизації $\Delta t = 0.1мс$

2. Знайти найбільше, найменше, середнє та середньоквадратичне значення сигналу.

3. Обчислити першу та другу похідну сигналу у часі та зберегти у окремих масивах.

4. Обчислити значення сигналу після дії рекурсивного фільтра та зберегти у окремому масиві.

5. Знайти усі локальні мінімуми або максимуми сигналу відповідно до номеру варіанту.

6. Знайти усі точки перетину сигналу із рівнем відповідно до номеру варіанту.

7. Розраховані значення похідних та обробленого фільтром сигналу зберегти у окремих файлах.

8. За допомогою *Matlab*, *Octave*, або іншого засобу побудувати графіки:

a. Вхідного сигналу

b. Першої похідної

c. Другої похідної

d. Обробленого фільтром сигналу

Таблиця 1 – Каталог вхідних даних

№	a_0	a_1	a_2	b_0	b_1	b_2	Шукати	Рівень
1	1	-0,6208	0,2405	0,1549	0,3099	0,1549	максимуми	0,0001469335
2	1	-0,8094	0,2898	0,1201	0,2402	0,1201	мінімуми	0,0001081809
3	1	-0,9405	0,3325	0,098	0,196	0,098	мінімуми	-0,0001348491
4	1	-0,7192	0,2645	0,1363	0,2727	0,1363	максимуми	0,0001318818
5	1	-0,8259	0,2948	0,1172	0,2345	0,1172	максимуми	-0,0001235644
6	1	-0,6963	0,2586	0,1406	0,2812	0,1406	максимуми	-0,0001006276
7	1	-0,8675	0,3079	0,1101	0,2202	0,1101	максимуми	0,0001400368
8	1	-0,9617	0,3401	0,0946	0,1892	0,0946	мінімуми	0,0001196426
9	1	-0,7221	0,2653	0,1358	0,2716	0,1358	максимуми	0,0001249366
10	1	-0,7471	0,272	0,1312	0,2624	0,1312	мінімуми	0,000137853
11	1	-0,7101	0,2622	0,138	0,276	0,138	мінімуми	-0,0001370957
12	1	-0,7611	0,2759	0,1287	0,2574	0,1287	мінімуми	-0,0001070066
13	1	-0,9173	0,3244	0,1018	0,2036	0,1018	максимуми	-0,0001051222
14	1	-0,6559	0,2487	0,1482	0,2964	0,1482	максимуми	-0,0001114828
15	1	-0,8179	0,2924	0,1186	0,2373	0,1186	мінімуми	0,0001225241
16	1	-0,78	0,2812	0,1253	0,2506	0,1253	максимуми	-0,0001433019
17	1	-0,8341	0,2973	0,1158	0,2316	0,1158	максимуми	0,0001205797
18	1	-0,9243	0,3268	0,1006	0,2013	0,1006	мінімуми	0,0001055299
19	1	-0,7972	0,2862	0,1223	0,2445	0,1223	мінімуми	-0,0001348439
20	1	-0,7277	0,2668	0,1348	0,2695	0,1348	мінімуми	0,0001227616
21	1	-0,7917	0,2846	0,1232	0,2465	0,1232	максимуми	-0,0001108533
22	1	-0,7265	0,2665	0,135	0,27	0,135	максимуми	0,0001290797
23	1	-0,8363	0,298	0,1154	0,2309	0,1154	мінімуми	0,0001037076
24	1	-0,6387	0,2446	0,1515	0,303	0,1515	максимуми	0,0001147956
25	1	-0,8591	0,3052	0,1115	0,2231	0,1115	мінімуми	0,0001468844

Зміст звіту

Звіт слід оформляти в електронному вигляді.

Він повинен містити такі пункти:

1. Мета роботи.

2. Завдання на виконання лабораторної роботи.

3. Лістинг коду програми та результати її виконання.

- Код програми необхідно подати у форматі тексту.
- Результати роботи програми слід навести у вигляді скріншотів або текстового виводу.

4. Висновки.

- Висновки мають бути індивідуальними та відображати сутність виконаної роботи.

Контрольні питання:

1. Які переваги має динамічне виділення пам'яті у разі обробки сигналів?
2. Як обчислюється перша та друга похідні дискретизованого сигналу?
3. Що таке рекурсивний фільтр, і які його основні параметри?
4. Які умови визначення локальних максимумів та мінімумів у масиві даних?
5. Як здійснюється пошук точок перетину сигналу з заданим рівнем?

Лабораторна робота № 2

Статистична обробка одновимірного масиву

Мета роботи: Ознайомлення з методами статистичної обробки одновимірного масиву даних. Закріплення навичок роботи з масивами у мовах програмування та вивчення алгоритмів розрахунку основних статистичних характеристик. Формування практичних навичок побудови інтервальних рядів і гістограм.

Короткі теоретичні відомості

Масив – вказівник на позицію у пам`яті, за якою контекстно передбачається послідовне розташування декількох полів вказаного типу. Масиви бувають статичні та динамічні [2]. Кількість полів статичного масиву має бути задана константно на етапі компіляції. Розмір динамічного масиву визначається під час виділення пам`яті. Для посилання на певний елемент масиву використовують оператор *[]*

int a[10]; a[5] = 3; що еквівалентно **(a+5*sizeof(int)) = 3;*

Фактично, змінна, яка представляє масив, є вказівником на його перший елемент.

Прийоми вичитування регулярного файлу у масив обговорені у ЛР №1.

За необхідності передати масив до функції, окремо передають вказівник та окремо – розмір масиву. Багатовимірні масиви розглядають, як масиви масивів меншої мірності.

Вважаючи масив значеннями випадкової величини (наприклад результатами вимірювання), можна знайти його основні статистичні характеристики:

- Найбільше та найменше значення.
- Середнє арифметичне значення: $\overline{x} = \frac{1}{n} \sum_{i=0}^{n-1} x_i$.
- Середнє гармонічне значення: $\overline{x}_{\text{сер гарм}} = \frac{n}{\sum_{i=0}^{n-1} \frac{1}{x_i}}$.

- Середнє квадратичне значення: $\bar{x}_{\text{сер кв}} = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} x_i^2}$.
- Середнє геометричне значення: $\bar{x}_{\text{сер геом}} = \sqrt[n]{\prod_{i=0}^{n-1} x_i}$.
- Середньоквадратичне відхилення: $S = \frac{1}{n}\sum_{i=0}^{n-1}\left(x_i - \bar{x}\right)^2$.

Інтервальним рядом для масиву *{N}* називають такий масив розміром *M*, що його елементи дорівнюють кількості елементів *{N}*, які потрапляють до відповідного інтервалу значень довжиною *(max{N} – min{N})/M*.

Сума елементів інтервального ряду дорівнює кількості елементів вихідного масиву.

Після обчислення значень інтервального ряду можна визначити інтервальні характеристики масиву:

Мода: $M_o = x_0 + h\frac{y_m - y_{m-1}}{\left(y_m - y_{m-1}\right) - \left(y_m - y_{m+1}\right)}$,

де x_0 – нижня границя модального інтервалу; h - ширина інтервалу; y_m – значення у модальному інтервалі; y_{m-1} та y_{m+1} – значення у попередньому та наступному інтервалах.

Модальним називається інтервал ряду із найбільшим значенням.

Медіана: $M_e = x_0 + h\frac{\frac{\sum_{i=0}^{n-1} y_i}{2} - \sum_{i=0}^{m-1} y_i}{y_m}$,

де n – кількість елементів ряду; m – номер медіанного інтервалу; $y_m - y_m$ - значення у медіанному інтервалі.

Медіанним називають інтервал, у якому знаходиться середина діапазону значень ряду, тобто $y_i/2$

Графічне зображення інтервального ряду називають гістограмою (див. рис. 1).

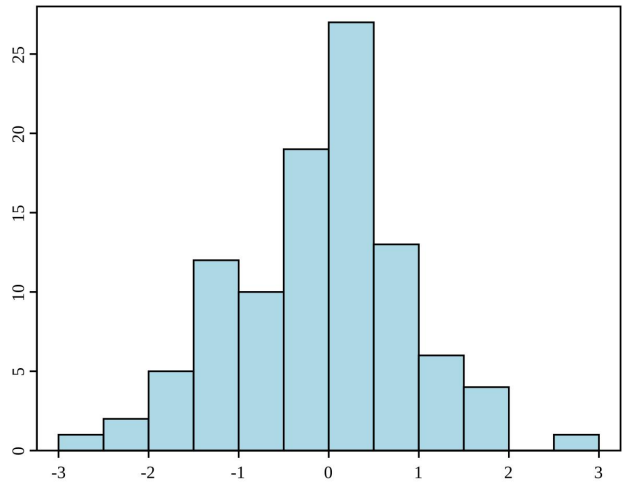


Рисунок 1 – Гістограма

Для побудови гістограми програмно слід задати масштаб осі ординат – значення інтервального ряду можуть бути будь якими, а кількість знакомісць на екрані завжди обмежена, тому алгоритм має бути таким:

– Знайти найбільше значення інтервального ряду

– Визначити максимальну допустиму висоту стовпчика на екрані у знаках

– Розділити кожне значення інтервального ряду на нормуючий коефіцієнт – відношення максимального значення у ряду на максимальну висоту стовпчика. Отримані значення округлити до цілих.

– Побудувати гістограму як таблицю символів, де у кожній клітинці знаходиться або пробіл, або символ, що складає стовпчик, наприклад прямокутники з ASCII кодами 178 чи 219.

Програма роботи:

1. Обрати файл (*l2vXX.txt*, де *XX* – номер варіанту) з каталогу вхідних даних для лабораторних робіт відповідно до номеру варіанту (див. табл. 2) та вичитати його у одномірний динамічний масив типу *int*.

2. Для масиву обчислити та надрукувати значення:

a. Розміру масиву

b. Максимуму

c. Мінімуму

d. Середнього значення за формулою відповідно до варіанту

e. Середньоквадратичного відхилення

3. Задати статично масив для інтервального ряду типу *int* розміром відповідно до номеру варіанту.

4. Написати функцію, що приймає масив даних, його розмір, масив інтервалів, його розмір, та обчислює значення елементів інтервального ряду. Викликати її з *main*.

5. Обчислити та надрукувати значення медіани та моди інтервального ряду.

6. Написати функцію, що приймає масив інтервального ряду та його розмір і друкує нормовану гістограму зі вертикальними стовпчиками. Викликати її з *main*.

Зміст звіту

Звіт слід оформляти в електронному вигляді.

Він повинен містити такі пункти:

1. Мета роботи.

2. Завдання на виконання лабораторної роботи.

3. Лістинг коду програми та результати її виконання.

- Код програми необхідно подати у форматі тексту.
- Результати роботи програми слід навести у вигляді скріншотів або текстового виводу.

4. Висновки.

- Висновки мають бути індивідуальними та відображати сутність виконаної роботи.

Таблиця 2 – Каталог вхідних даних

Номер варіанту	**Тип середнього**	**Кількість інтервалів ряду**
1	геометричне	9
2	гармонічне	8
3	арифметичне	8
4	гармонічне	6
5	квадратичне	9
6	гармонічне	6
7	квадратичне	6
8	арифметичне	9
9	геометричне	8
10	геометричне	7
11	арифметичне	8
12	гармонічне	8
13	арифметичне	7
14	арифметичне	6
15	геометричне	6
16	гармонічне	7
17	квадратичне	6
18	арифметичне	8
19	геометричне	6
20	геометричне	7
21	квадратичне	7
22	квадратичне	7
23	арифметичне	9
24	квадратичне	9
25	гармонічне	9

Контрольні питання:

1. Що таке масив у мовах програмування? Які його основні характеристики?

2. У чому відмінність між статичним та динамічним масивом?

3. Як передати масив у функцію?

4. Які основні статистичні характеристики можна обчислити для вибірки чисел?

5. Як обчислюється середнє арифметичне, геометричне, гармонічне та квадратичне значення?

6. Що таке інтервальний ряд та як його побудувати?

7. Як знайти моду та медіану для інтервального ряду?

8. Що таке гістограма та як її побудувати програмно?

9. Який алгоритм нормалізації даних для гістограми?

10. Як обрати оптимальну кількість інтервалів для аналізу розподілу даних?

Лабораторна робота № 3

Похибки обчислень з плаваючою комою

Мета роботи: Ознайомлення з похибками обчислень з числами з плаваючою комою, вивчення особливостей їх представлення у пам'яті комп'ютера відповідно до стандарту IEEE 754. Дослідження впливу накопичення похибок на точність обчислень та отримання практичних навичок у визначенні та оцінці похибок арифметичних операцій.

Короткі теоретичні відомості

Для виконання цієї роботи, необхідно повторити тему «Двійковий запис реальних чисел» та «Похибки обчислень з реальними числами».

Число з плаваючою комою (точкою) – форма представлення дійсних чисел, в якій число зберігається у формі мантиси і показника ступеня. Отже число з плаваючою комою має фіксовану відносну точність і мінливу абсолютну. Найбільш часто використовуване уявлення затверджено в стандарті ***IEEE* 754**. Реалізація математичних операцій з числами з плаваючою комою в обчислювальних системах може бути як апаратна, так і програмна.

У деяких країнах, переважно англомовних або таких, що зазнали впливу англійської мови, десятковий роздільник у числах представлений крапкою, тому там використовується термін «***плаваюча крапка***» (*floating point*). В Україні ж традиційно ціла та дробова частини числа відокремлюються комою, що історично зумовило вживання терміну «***плаваюча кома***». Однак у сучасній технічній літературі та документації можна зустріти обидва варіанти.

Назва «***плаваюча кома***» походить від того, що кома у позиційному поданні числа (десяткова кома, або, для комп'ютерів, двійкова кома – далі за текстом просто кома) може бути поміщена де завгодно щодо цифр у рядку. Це положення коми вказується окремо у внутрішньому поданні. Отже, подання числа у формі з плаваючою комою може розглядатися як комп'ютерна

реалізація експоненціального запису чисел.

Перевага використання представлення чисел у форматі з плаваючою комою над виставою у форматі з фіксованою комою (і цілими числами) полягає у тому, що можна використовувати істотно більший діапазон значень за незмінної відносної точності. Наприклад, у формі з фіксованою комою число, що займає 8 розрядів у цілій частині і 2 розряду після коми, може бути представлено у вигляді: **123456,78; 8765,43; 123,00** і так далі. У свою чергу, у форматі з плаваючою комою (у тих же 8 розрядах) можна записати числа.

1,2345678; 1234567,8; 0,000012345678; 12345678000000000 і так далі, але для цього необхідно дворозрядне додаткове поле для запису показників ступеня 10 від 0 до 16 10, а проте загальне число розрядів складе 8 + 2 = 10.

Швидкість виконання комп'ютером операцій з числами, представленими у формі з плаваючою комою, вимірюється у мегафлопсах (від англ. ***FLOPS*** – ***число операцій з плаваючою комою в секунду***), гігафлопсах і так далі, і є однією з основних одиниць вимірювання швидкодії обчислювальних систем [3].

Структура числа

Число з плаваючою комою складається з:

- Мантиси (що виражає значення числа без урахування порядку)
- Знака мантиси (що вказує на негативні чи позитивні числа)
- Порядку (виражає ступінь підстави числа, на яке множиться мантиса)
- Знака порядку

Нормальна форма і нормалізована форма

Нормальною формою числа з плаваючою комою називається така форма, в якій мантиса (без урахування знака) знаходиться на полуінтервалі **[0; 1)** ($0 \quad a < 1$). Число з плаваючою комою, що знаходиться не у *нормальній формі,* втрачає точність у порівнянні з *нормальною формою.* Така форма запису має

недолік: деякі числа записуються неоднозначно (наприклад, **0,0001** можна записати у 4 формах – **$0{,}0001 \cdot 10^{0}$, $0{,}001 \cdot 10^{-1}$, $0{,}01 \cdot 10^{-2}$, $0{,}1 \cdot 10^{-3}$**), тому поширена (особливо в інформатиці) також інша форма запису – ***нормалізована,*** в якій мантиса десяткового числа приймає значення від 1 (включно) до 10 (не включно), а мантиса двійкового числа приймає значення від 1 (включно) до 2 (не включно) ($1 \quad a < 2$). У такій формі будь-яке число (крім 0) записується єдиним чином. Недолік полягає у тому, що у такому вигляді неможливо уявити 0, тому представлення чисел в інформатиці передбачає спеціальну ознаку (біт) для числа 0.

Так як старший розряд (ціла частина числа) мантиси двійкового числа (крім 0) у ***нормалізованому*** вигляді дорівнює «1», то у разі запису мантиси числа в ЕОМ старший розряд можна не записувати, що і використовується у стандарті *IEEE* 754. У позиційних системах числення з основою більшою, ніж 2 (в трійковій, четвірковій та ін), цієї властивості немає.

В обчислювальних машинах показник ступеня прийнято відокремлювати від мантиси буквою «Е» (*exponent*). Наприклад, число **$1{,}528535047 \cdot 10^{-25}$** в більшості мов програмування високого рівня записується як **1.528535047E-25**.

Способи машинної реалізації.

Існує кілька способів того, як рядки з цифр можуть представляти числа [4]:

Найбільш поширений шлях подання значення числа з рядка з цифрами – у вигляді цілого числа – кома (*radix point*) за замовчуванням знаходиться у кінці рядка.

Загалом математичному уявленні рядок з цифр може бути як завгодно довгою, а положення коми позначається шляхом явною запису символу комою (або, на Заході, точки) у потрібному місці.

У системах з поданням чисел у форматі з фіксованою комою існує певна умова щодо положення коми. Наприклад, у рядку з 8 цифр умова може наказувати положення коми в середині запису (між 4-ю і 5-ю цифрою). Отже,

рядок «00012345» означає число 1,2345 (нулі зліва завжди можна відкинути).

У експоненціальному записі використовують стандартний (*нормалізований*) вид представлення чисел. Число вважається записаним у стандартному (*нормалізованому*) вигляді, якщо воно записане у вигляді aq^n, де a, називають мантисою, таке, що, $1 \quad a < q$, де n – ціле, називається показник ступеня та q – ціле, основа системи числення (на практиці це зазвичай 10). Тобто у мантиси кома поміщається відразу після першої значущої (не дорівнює нулю) цифри, рахуючи зліва направо, а подальший запис дає інформацію про дійсне значення числа. Наприклад, період обігу (на орбіті) супутника планети Юпітера Іо, який дорівнює 152853,5047 с, у стандартному вигляді можна записати як 1,528535047 10^5с. Побічним ефектом обмеження на значення мантиси є те, що у такому записі неможливо зобразити число 0.

Запис у формі з плаваючою комою схожий на запис чисел у стандартному вигляді, але мантиса і експонента записуються роздільно. Мантиса записується у *нормалізованому* форматі – з фіксованою комою, яка мається на увазі після першої значущої цифри. Повертаючись до прикладу з Іо, запис у формі з плаваючою комою буде 1528535047 з показником 5. Це означає, що записане число у 10^5 разів більше числа 1,528535047, тобто для отримання фактичного значення числа кома зсувається на 5 розрядів вправо. Однак запис у форматі з плаваючою комою переважно використовується в електронному поданні чисел, де застосовується двійкова система числення замість десяткової. Крім того, в двійковому запису мантиса зазвичай денормалізована, тобто кома мається на увазі до першої цифри, а не після, і цілої частини взагалі не мається на увазі – так з'являється можливість і значення 0 зберегти природним чином. Отже, десяткова 9 у двійковому поданні з плаваючою комою буде записана як мантиса +1001000 ... 0 і показник +0 ... 0100. Звідси, наприклад, біди з двійковим поданням чисел типу однієї десятої (0,1), для якої двійкове подання мантиси виявляється періодичної двійковій дробом – за аналогією з 1/3, яку не можна кінцевим кількістю цифр записати в десятковій системі числення.

Запис числа у формі з плаваючою комою дозволяє робити обчислення над широким діапазоном величин, поєднуючи фіксована кількість розрядів і точність. Наприклад, у десятковій системі надання чисел з плаваючою комою (3 розряду) операцію множення, яку ми б записали як

$0{,}12 \cdot 0{,}12 = 0{,}0144$

у нормальній формі представляється у вигляді

$(1{,}20 \cdot 10^{-1}) \; (1{,}20 \cdot 10^{1}) = (1{,}44 \; 10^{-2})$.

У форматі з фіксованою комою ми б отримали вимушене округлення

$0{,}120 \cdot 0{,}120 = 0{,}014$.

Ми втратили крайній правий розряд числа, так як даний формат не дозволяє комі "плавати" у запису числа.

Діапазон чисел, які можуть бути представлені у форматі з плаваючою комою

Діапазон чисел, які можна записати даними способом, залежить від кількості біт, відведених для представлення мантиси і показника. На звичайній

32-бітної обчислювальній машині, що використовує подвійну точність (64 біта), мантиса становить 1 біт знак + 52 біта, показник – 1 біт знак + 10 біт. Отже отримуємо діапазон точності приблизно від $4{,}94 \cdot 10^{-324}$ до $1{,}79 \cdot 10^{308}$ (від $2^{-52} \; 2^{-1022}$ до $\sim 1 \; 2^{1024}$). Пара значень показника зарезервована для забезпечення можливості подання спеціальних чисел. До них відносяться значення *NaN* (*Not a Number*, не число) і + /-*INF* (*Infinity*, нескінченність), які утворюються у результаті операцій типу поділу на нуль нуля, позитивних і негативних чисел. Також сюди потрапляють денормалізовані числа, у яких мантиса менше одиниці. У спеціалізованих пристроях (наприклад *GPU*) підтримка спеціальних чисел часто відсутня. Існують програмні пакети, в яких

обсяг пам'яті виділений під мантису і показник задається програмно, і обмежується лише обсягом доступної пам'яті ЕОМ (див. табл. 3) [5].

Таблиця 3 – Каталог вхідних даних

Точність	**Одинарна**	**Подвійна**	**Розширена**
Розмір (байти)	4	8	10
Число десяткових знаків	7	15	19
Найменше значення (> 0), denorm	$1{,}4\cdot10^{-45}$	$5{,}0\cdot10^{-324}$	$1{,}9\cdot10^{-4\,951}$
Найменше значення (> 0), normal	$1{,}2\cdot10^{-38}$	$2{,}3\cdot10^{-308}$	$3{,}4\cdot10^{-4\,932}$
Найбільше значення	$3{,}4\cdot10^{+38}$	$1{,}7\cdot10^{+308}$	$1{,}1\cdot10^{+4932}$
Поля	*SEF*	*SEF*	*SEIF*
Розміри полів	1-8-23	1-11-52	1-15-1-63

S – знак, *E* – показник ступеня, *I* – ціла частина, *F* – дробова частина

Так само, як і для цілих, знаковий біт – старший.

Машинний іпсилон

На відміну від чисел з фіксованою комою, сітка чисел, які здатна відобразити арифметика з плаваючою комою, нерівномірна: вона густіша для чисел з малими порядками і більш рідка - для чисел з великими порядками. Але відносна похибка запису чисел однакова і для малих чисел, і для великих. Тому можна ввести поняття машинної епсилон.

Машинним іпсилон називається найменше позитивне число ε таке, що $1 \oplus \varepsilon \neq 1$ (знаком $\oplus$ позначено машинне складання). На практиці це означає, що машинна арифметика для даного типу даних не розрізняє числа *a* та *b* такі, що $1 < a/b < 1+\varepsilon$

Програма роботи:

1. Написати програму, яка обчислює у типі такі *float* дробі:
    a. 1/3
    b. 1/33
    c. 1/333
    d. 1/3333
    e. 1/33333
2. Обчислити ті самі дробі у типі *long double* та обчислити похибку округлення. Порівняти та пояснити результат.
3. Написати програму, яка виконує такі дії із змінною типу *float*:
    a. Додає 0.1 десять разів,
    b. Додає 0.01 сто разів,
    c. Додає 0.001 тисячу разів
    d. Додає 0.000001 мільйон разів
4. Обчислити похибку додавання. Порівняти та пояснити результат.
5. Написати програму, яка обчислює такі послідовності (лімітом є log 2), використовуючи змінні типу *float*:

$$а)\quad 1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\frac{1}{5}-\cdots-\frac{1}{10000}$$

$$б)\quad -\frac{1}{10000}+\frac{1}{9999}-\frac{1}{9998}+\cdots-\frac{1}{2}+1$$

$$в)\quad 1+\frac{1}{3}+\frac{1}{5}+\cdots\frac{1}{9999}\ -\ \frac{1}{2}+\frac{1}{4}+\frac{1}{6}+\cdots\frac{1}{10000}$$

$$г)\quad \frac{1}{9999}+\frac{1}{9997}+\cdots\frac{1}{3}+1\ -\ \frac{1}{10000}+\frac{1}{9998}+\cdots\frac{1}{4}+\frac{1}{2}$$

6. Порівняти та пояснити результат.
7. Написати функцію для обчислення машинного іпсилон. Обчислити машинний іпсилон для типу *float*.

Зміст звіту

Звіт слід оформляти в електронному вигляді.

Він повинен містити такі пункти:

1. Мета роботи.

2. Завдання на виконання лабораторної роботи.

3. Лістинг коду програми та результати її виконання.

- Код програми необхідно подати у форматі тексту.
- Результати роботи програми слід навести у вигляді скріншотів або текстового виводу.

4. Висновки.

- Висновки мають бути індивідуальними та відображати сутність виконаної роботи.

Контрольні питання:

1. Що таке числа з плаваючою комою?
2. Які основні компоненти числа з плаваючою комою?
3. У чому полягає відмінність між нормальною та нормалізованою формою представлення чисел?
4. Який формат представлення чисел передбачено стандартом IEEE 754?
5. Як впливає розрядність представлення чисел на точність обчислень?
6. Що таке машинний епсилон? Як його визначити?
7. Які причини виникнення похибок у числових обчисленнях?
8. Чому операції додавання малих чисел до великих можуть призводити до втрати точності?
9. Які методи мінімізації похибок обчислень існують?
10. Як впливає порядок виконання арифметичних операцій на результат обчислень?

Лабораторна робота № 4

Операції з двовимірними масивами

Мета роботи: Ознайомлення з операціями над двовимірними масивами, закріплення навичок роботи з динамічними структурами даних у мовах програмування. Вивчення методів сортування, транспонування та перетворення матриць, а також оптимізації роботи з пам’яттю у разі виділення та звільнення динамічних масивів.

Короткі теоретичні відомості

Для виділення пам`яті під час виконання програми, використовують функцію *malloc*, яка приймає розмір потрібного блоку, та повертає вказівник на нього або *NULL*.

Двомірні масиви описують подвійними вказівниками (вказівниками на вказівники), наприклад *int **a;* Пам`ять двовимірного масиву виділяють послідовно – для масиву вказівників (рядків) та для кожного рядка окремо.

Для зміни розміру виділеного блоку використовують функцію *realloc*.

Пам`ять, яка була виділена функцією *malloc*, повинна бути звільнена функцією *free*.

Для того, щоб завантажити з файлу матрицю регулярної структури, потрібно спочатку визначити кількість стовпчиків матриці. Найпростіше зробити це отримавши перший рядок файлу за допомогою *fgets*. У отриманому масиві символів слід підрахувати кількість символів належних до числа (цифри, плюс, мінус, точка), перед якими стоїть не належний до числа символ, що відповідає кількості чисел у рядку.

Наприклад:

```
char buf[255];

char c;
```

```
int i,n,m,k;
int **a;
...
fgets(buf,255,fd);
i = 0; k = 0; m = 0;
do{
  c = buf[i++];
  if((c>47 && c<58)||c==43||c==45||c==46){
        if(!k){
              k = 1;
              m++;
            }
      }else k = 0;
}while(i<255 && c > 0);
```

Далі слід перевідкрити файл спочатку та почати завантажувати двовимірний масив додаючи до нього нові рядки:

```
  rewind(fd);
  n = 1; k = 0;
  a = (int**)malloc(n*sizeof(int*));
  a[n-1] = (int*)malloc(m*sizeof(int));
  while(!feof(fd)){
```

```
 fscanf(fd,"%d",&a[n-1][k++]);
 if(k == m){
                a = (int**)realloc(a,(++n)*sizeof(int*));
        a[n-1] = (int*)malloc(m*sizeof(int));
        k = 0;
       }
}
if(k == 0){
 free(a[n-1]);
 n--;
}
```

Далі можна використовувати двовимірний масив *a* розмірами *n* на *m*.

Для перестановки рядків динамічно виділеної матриці, достатньо поміняти місцями вказівники на них. Перестановку стовпчиків доведеться робити циклом.

Програма роботи:

1. Обрати файл з каталогу вхідних даних для лабораторних робіт відповідно до номеру варіанту такий самий, як для лабораторної роботи № 2. Завантажити файл у динамічний двомірний масив типу *int*, визначити його розміри {*N*, *M*}.

2. Виділити два окремі одновимірні масиви типу *int* розмірами відповідно до кількості рядків та стовпчиків у матриці. Розрахувати та записати у масиви суми квадратів елементів кожного рядка та кожного стовпчика матриці.

3. Надрукувати отриману матрицю та суми квадратів.

4. Розмістити рядки матриці у порядку зростання сум квадратів їх елементів. Вивести результат.

5. Розмістити стовпчики матриці у порядку зростання сум квадратів їх елементів. Вивести результат.

6. Виділити пам`ять під двовимірний масив цілих чисел розміром [*M* x *N*] та записати у нього транспоновану матрицю. Вивести результат.

7. Обрізати транспоновану матрицю до квадратної, видаливши з неї зайві останні рядки.

8. У квадратній матриці переставити рядки так, щоб на головній діагоналі опинилися найменші чи найбільші (відповідно до варіанту) серед елементів відповідного стовпчика нижче головної діагоналі. Вивести результат.

9. Виконати перетворення матриці відповідно до номеру варіанту (див. табл. 4). Вивести результат.

Таблиця 4 – Каталог вхідних даних

№ варіанту	Сортувати елементи у стовпчиках за	Перетворення
1	спаданням	Знайти найбільший елемент матриці. Поміняти його рядок та стовпчик з першими.
2	спаданням	Помножити матрицю на верхню трикутну заповнену одиницями
3	спаданням	Відсортувати рядки матриці за спаданням елементів побічної діагоналі
4	спаданням	Обернути матрицю за годинниковою стрілкою
5	зростанням	Поміняти місцями 2й та 4й квадранти матриці. Для непарного розміру центральний рядок та стовпчик не змінювати.
6	спаданням	Поміняти місцями 1й та 3й квадранти матриці. Для непарного розміру центральний рядок та стовпчик не змінювати.
7	зростанням	Відсортувати стовпчики матриці за зростанням елементів головної діагоналі
8	зростанням	Відобразити матрицю відносно горизонтальної осі.

9	спаданням	Помножити матрицю на транспоновану
10	спаданням	Піднести матрицю до куба

Продовження тблиці 4

11	спаданням	Відобразити матрицю відносно головної діагоналі
12	зростанням	Відсортувати рядки матриці за спаданням елементів головної діагоналі
13	спаданням	Відсортувати рядки матриці за зростанням елементів головної діагоналі
14	зростанням	Відобразити матрицю відносно вертикальної осі.
15	зростанням	Помножити матрицю на нижню трикутну заповнену одиницями
16	зростанням	Поміняти місцями верхній та нижній трикутники матриці
17	спаданням	Видалити з матриці рядки та стовпчики, на перетині яких є число, кратне 3.
18	зростанням	Відсортувати стовпчики матриці за спаданням елементів головної діагоналі
19	зростанням	Обернути матрицю проти годинникової стрілки
20	спаданням	Поміняти місцями лівий та правий трикутники матриці
21	зростанням	Відобразити матрицю відносно побічної діагоналі
22	зростанням	Відсортувати рядки матриці за зростанням елементів побічної діагоналі
23	зростанням	Виконати загострення матриці додавши до кожного елемента найбільшу різницю між ним та усіма його сусідами.
24	спаданням	Знайти найменший елемент матриці. Поміняти його рядок та стовпчик з першими.
25	зростанням	Виконати згладжування матриці замінивши кожен її елемент на середнє арифметичне усіх його сусідів з округленням до цілого.

Зміст звіту

Звіт слід оформляти в електронному вигляді.

Він повинен містити такі пункти:

1. Мета роботи.

2. Завдання на виконання лабораторної роботи.

3. Лістинг коду програми та результати її виконання.

- Код програми необхідно подати у форматі тексту.
- Результати роботи програми слід навести у вигляді скріншотів або текстового виводу.

4. Висновки.

- Висновки мають бути індивідуальними та відображати сутність виконаної роботи.

Контрольні питання:

1. Як у мовах програмування оголошуються та використовуються двовимірні масиви?
2. Які способи виділення пам’яті для двовимірних масивів існують?
3. Як звільняти динамічно виділену пам’ять для двовимірного масиву?
4. Як зчитати двовимірний масив із файлу?
5. Як реалізувати сортування рядків і стовпців у матриці?
6. У чому полягає транспонування матриці? Як його реалізувати програмно?
7. Які існують алгоритми обробки двовимірних масивів?
8. Як реалізувати перестановку рядків і стовпців у матриці?
9. Які методи перетворення матриці використовуються у числових обчисленнях?
10. Як оптимізувати використання пам’яті при роботі з великими двовимірними масивами?

Лабораторна робота № 5

Операції з лінійними списками

Мета роботи: Ознайомлення з операціями над лінійними списками, вивчення основних структур даних, що використовують вказівники, та закріплення навичок роботи з динамічною пам'яттю. Формування практичних навичок роботи з однозв'язними списками, виконання основних операцій: додавання, видалення, перестановка елементів, а також реалізація операцій над векторами у вигляді спискових структур.

Короткі теоретичні відомості

Вказівник – тип даних, який описує адресу змінної іншого типу у пам`яті. Розмір вказівника залежить лише від того, у ближній чи дальній зоні пам`яті розташована змінна. Тип даних, на які вказує вказівник визначає розмір блоку пам`яті, який записується або читається за вказаною адресою (напр. *int*, char*, far double**) [6].

До вказівників застосовуються дві основні операції:

- адресування: *int a; int* b; b = &a;*
- розадресування: *int a; int* b = &a; *b = 3;*

Структура – тип даних, який складається з кількох полів різних типів, розташованих у певному порядку [4]. Розмір структури дорівнює сумі розмірів полів. Поля структури мають власні імена. Доступ до полів здійснюється через точку для змінної типа структура та через знак -> для вказівника на структуру.

Наприклад:

```
struct pair { int first; int second; };

struct pair x, *p;

x.first = 1; x.second = 2;

p = &x; p->first = 3;
```

Список – це структура, що містить один чи більше вказівників на таку саму структуру (сусідів по списку). Списки бувають:

- лінійні однозв'язані – містять один вказівник на наступний елемент списку
- лінійні двозв'язані – містять два вказівники на наступний та попередній елемент списку
- лінійні кільцьові – вказівники першого та останнього елементу об'єднані
- дерева – містять кілька вказівників на сусідні (похідні) елементи дерева.

Списки передбачають такі основні операції:

- ітерування – переміщення окремо виділеного вказівника між елементами списку шляхом присвоювання йому значення вказівника на сусіда відповідного елементу списку
- додавання елементу у кінець, початок чи середину списку – відбувається шляхом виділення пам'яті під нову структуру елементу і присвоювання вказівника на неї відповідному вказівнику на сусіда елемента списку
- вилучення елементу з початку, кінця чи середини – відбувається шляхом переприсвоювання вказівників на сусідів сусідніх елементів списку, та звільнення пам'яті елементу, що видаляється
- перестановка елементів у списку – відбувається шляхом переприсвоювання вказівників на сусідів відповідно до бажаного порядку

Приклади для лінійного однозв'язного списку:

```
typedef struct _list { double data; struct _list *next; } list;

list *head = (list*)malloc(sizeof(list));

head->data = 0;

// тут вказівник head зберігає адресу першого елеемнту списку,

// для однозв'язного списку рухатись можна тільки у одному напрямку
```

```
//// додавання п’яти елементів:
list *x = head;
for(i=0; i<5; i++){
        x->next = (list*)malloc(sizeof(list));
        x->next->data = i;
        x = x->next;
}
//// ітерування списком та друк елементів
x = head;
while(x){
  printf(“%f\n”,x->data);
  x = x->next;
}
//// обмін другого та третього елементів
x = head->next->next;
list *y = head;
y->next->next = x->next;
x->next = y->next;
y->next = x;
//// видалення третього елементу
x = head->next->next;
y = x->next;
x->next = y->next;
free(y);
```

Програма роботи:

1. Описати структуру *vector*, що складається з реальних полів *x,y,z* та є елементом однозв'язного списку бо містить вказівник на наступний елемент.

2. Обрати за номером варіанту файл (*l5vXX.txt*, де *XX* – номер варіанту) з каталогу вхідних даних для лабораторних робіт №5.

3. Прочитати з файлу групи з трьох реальних числа, вважати їх координатами *x,y,z* та створювати та додавати до списку нові елементи з такими координатами.

4. Надрукувати елементи списку та їх підраховану кількість.

5. Скласти функцію, яка обчислює скалярний добуток двох векторів, переданих вказівниками, та повертає реальне значення.

6. Скласти функцію, яка обчислює кут між двох векторів, переданих вказівниками, та повертає реальне значення.

7. Скласти функцію, яка обчислює суму двох векторів, переданих вказівниками та записує у третій вектор, який приймає через вказівник.

8. Скласти функцію, яка обчислює векторний добуток двох векторів, переданих вказівниками та записує у третій вектор, який приймає через вказівник.

9. Скласти функцію, яка обчислює добуток вектора, поданого вказівником, на скаляр та записує результат у інший вектор, поданий вказівником. Якщо другий вектор не заданий (*NULL*) – записати результат у перший вектор.

10. Останні три функції повинні повертати той вказівник на структуру, у який вони записують результат, або *NULL*, якщо операція не виконана.

11. Виконати зі списком операцію відповідно до номеру варіанту (див. табл. 5) та надрукувати результат.

Таблиця 5 – Каталог вхідних даних

№ варіанту	Задача
1	Знайти два сусідніх вектора, кут між якими найменший
2	Знайти суму усіх векторів списку
3	Знайти векторний добуток першого та останнього елемента списку
4	Знайти скалярний добуток першого та останнього елемента списку
5	Знайти усі вектори, що є локальними максимумами за модулем. Надрукувати їх координати та номери.
6	Знайти найбільший та найменший за модулем вектори та їх скалярний добуток.
7	Знайти найбільший за модулем вектор, надрукувати його номер та координати
8	Знайти два сусідніх вектора, кут між якими найбільший
9	Знайти найбільший та найменший за модулем вектори та їх векторний добуток.
10	Знайти найменший за модулем вектор, надрукувати його номер та координати
11	Знайти усі вектори, що є локальними мінімумами за модулем. Надрукувати їх координати та номери.
12	Знайти добуток суми усіх векторів та скалярного добутку першого та останнього вектора
13	Знайти найменший за модулем вектор, надрукувати його номер та координати
14	Знайти скалярний добуток першого та останнього елемента списку
15	Знайти два сусідніх вектора, кут між якими найменший
16	Знайти усі вектори, що є локальними максимумами за модулем. Надрукувати їх координати та номери.

Продовження таблиці 5

17	Знайти найбільший за модулем вектор, надрукувати його номер та координати
18	Знайти добуток суми усіх векторів та скалярного добутку першого та останнього вектора
19	Знайти два сусідніх вектора, кут між якими найбільший
20	Знайти найбільший та найменший за модулем вектори та їх векторний добуток.
21	Знайти усі вектори, що є локальними мінімумами за модулем. Надрукувати їх координати та номери.
22	Знайти векторний добуток першого та останнього елемента списку
23	Знайти суму усіх векторів списку
24	Знайти найбільший та найменший за модулем вектори та їх скалярний добуток.
25	Знайти суму усіх векторів списку

Зміст звіту

Звіт слід оформляти в електронному вигляді.

Він повинен містити такі пункти:

1. Мета роботи.

2. Завдання на виконання лабораторної роботи.

3. Лістинг коду програми та результати її виконання.

- Код програми необхідно подати у форматі тексту.
- Результати роботи програми слід навести у вигляді скріншотів або текстового виводу.

4. Висновки.

- Висновки мають бути індивідуальними та відображати сутність виконаної роботи.

Контрольні питання:

1. Що таке вказівник і як він використовується у програмуванні?
2. Яка основна відмінність між масивами та списками?
3. Як працює операція розіменування вказівника?
4. Які існують типи лінійних списків?
5. Як здійснюється динамічне виділення пам’яті для елементів списку?
6. Як відбувається додавання та видалення елементів у лінійному списку?
7. Як знайти довжину лінійного списку?
8. Як можна реалізувати сортування елементів у списку?
9. Як реалізувати операції скалярного та векторного добутку двох векторів?
10. Які переваги та недоліки спискових структур у порівнянні з масивами?

Лабораторна робота № 6
Алгоритми сортування

Мета роботи: Ознайомитися з різними алгоритмами сортування, їх принципами роботи, характеристиками та ефективністю. Навчитися реалізовувати алгоритми сортування у вигляді функцій, аналізувати їхню швидкодію та використання пам’яті в залежності від структури вхідних даних. Провести порівняння обраних методів сортування за критеріями стійкості, природності та асимптотичної складності.

Короткі теоретичні відомості

Сортування масиву чи списку потребує визначення двох основних операцій – порівняння (визначення чи є обрана пара значень правильно відсортованою) та обміну (поміняти місцями два елемента). Велика кількість алгоритмів сортування спричинена тим, що тривіальні методи надто повільні на великих масивах і розробники постійно шукають оптимізації, що прискорюють алгоритм у специфічних випадках – частково відсортованих масивах з повторами, з відомими функціями розподілу чи при наявності апріорної інформації про множини, до яких належать елементи [7].

Для порівняння алгоритмів сортування використовують такі основні показники:

- стійкість – здатність не виконувати обмін однакових елементів
- природність – здатність ефективно обробляти вже відсортовані чи частково відсортовані масиви. Повністю природний алгоритм на відсортованому масиві робить n порівнянь та жодного обміну.
- асимптотика часу (швидкість, або обчислювальна складність) – оцінка кількості виконаних операцій, оцінена зверху (через О-велике) у гіршому, середньому та кращому випадках.

• пам’ять – кількість додатково витраченої на виконання алгоритму пам’яті віднесеної до розміру елементу що сортується.

Описи найбільш вживаних алгоритмів сортування.

Сортування бульбашкою / *Bubble sort*

Будемо йти за масивом зліва направо. Якщо поточний елемент перевищує наступний, міняємо їх місцями. Робимо так, поки масив не буде відсортований. Зауважимо, що після першої ітерації найбільший елемент буде знаходитися в кінці масиву, на правильному місці. Після двох ітерацій на правильному місці стоятимуть два найбільших елемента, і так далі. Очевидно, не більше ніж після n ітерацій масив буде відсортований. Отже, оцінка числа операцій у гіршому і середньому випадку – O (n_2), у кращому випадку – O (n).

Шейкерне сортування / Shaker sort

(Також відоме як сортування перемішуванням або коктейльне сортування). Зауважимо, що сортування бульбашкою працює повільно на тестах, в яких маленькі елементи стоять у кінці (їх ще називають «черепахами»). Такий елемент на кожному кроці алгоритму буде зрушуватися всього на одну позицію вліво. Тому будемо одночасно йти не тільки зліва направо, а й справа наліво. Будемо підтримувати два вказівника *begin* і *end*, що позначають, який відрізок масиву ще не впорядкований. На черговій ітерації у разі досягнення *end* віднімаємо з нього одиницю і рухаємося справа наліво, аналогічно, у разі досягнення *begin* додаємо одиницю і рухаємося зліва направо. Оцінка числа операцій у алгоритму така ж, як і у сортування бульбашкою, проте реальний час роботи краще.

Сортування гребінцем / *Comb sort*

Ще одна модифікація сортування бульбашкою. Для того, щоб позбутися від «черепах», будемо переставляти елементи, які стоять на відстані.

Зафіксуємо його і будемо йти зліва направо, порівнюючи елементи, які стоять на цій відстані, переставляючи їх, якщо необхідно. Очевидно, це дозволить «черепахам» швидко дістатися початку масиву. Оптимально спочатку взяти відстань рівну довжині масиву, а далі ділити її на деякий коефіцієнт, що дорівнює приблизно 1.247. Коли відстань стане дорівнювати одиниці, виконується сортування бульбашкою. У кращому випадку оцінка числа операцій дорівнює O($nlogn$), у гіршому – O(n_2).

Сортування вставками / *Insertion sort*

Створимо масив, в якому після завершення алгоритму буде розміщено результат. Будемо за чергою вставляти елементи з вихідного масиву так, щоб елементи у масиві-результаті завжди були відсортовані. Оцінка числа операцій у середньому і найгіршому випадку – O(n_2), у кращому – O(n). Для реалізації алгоритму зручно використати зв’язаний список з одним елементом даних та одним посиланням на наступний елемент. Тоді першим проходом за масивом для кожного елементу створюється елемент списку та вставляється у потрібне місте, другим проходом за списком формується відсортований масив.

Сортування Шелла / Shellsort

Використовуємо ту ж ідею, що і сортування з гребінцем, і застосуємо до сортування вставками. Зафіксуємо деяку відстань. Тоді елементи масиву розіб'ються на класи – в один клас потрапляють елементи, відстань між якими кратно зафіксованим віддалі. Відсортуємо сортуванням вставками кожен клас. На відміну від сортування гребінцем, невідомий оптимальний набір відстаней. Існує досить багато послідовностей з різними оцінками. Найпростішою є послідовність Шелла – перший елемент дорівнює довжині масиву, кожен наступний вдвічі менше попереднього. Оцінка числа операцій у гіршому випадку – O(n_2). Також застосовують послідовності Хіббарда, Седжвіка, Пратта та інші. Ви можете обрати послідовність за смаком. Послідовності

проріджування необхідно розраховувати до розміру масиву та застосовувати від більшого до меншого.

Сортування гнома / *Gnome sort*

Алгоритм схожий на сортування вставками. Підтримуємо вказівник на поточний елемент, якщо він більше попереднього або він перший - зміщуємо вказівник на позицію вправо, інакше змінюємо поточний і попередній елементи місцями і зміщується вліво.

Сортування вибором / *Selection sort*

На черговій ітерації будемо знаходити мінімум у масиві після поточного елемента і міняти його з ним, якщо треба. Отже, після *i*-ої ітерації перші і елементів стоятимуть на своїх місцях. Оцінка числа операцій: O(n_2) у кращому, середньому і найгіршому випадку. Потрібно відзначити, що це сортування можна реалізувати двома способами - зберігаючи мінімум і його індекс або просто переставляючи поточний елемент з даним, якщо вони стоять в неправильному порядку. Перший спосіб виявляється дещо швидшим.

Швидке сортування / *Quicksort*

Виберемо деякий опорний елемент. Після цього перекинемо всі елементи, менші його, ліворуч, а більші – направо. Рекурсивно розділяємо так само ліву та праву частини масиву аж доки довжина підмасиву на стане меншою 2. В результаті отримаємо відсортований масив, оскільки кожен елемент менше опорного опинився лівіше кожного більшого. Оцінка числа операцій: O(*nlogn*) у середньому і кращому випадку, O(n_2). Найгірша оцінка досягається при невдалому виборі опорного елемента.

Сортування злиттям / *Merge sort*

Сортування, засноване на парадигмі «розділяй і володарюй». Розділимо масив навпіл, рекурсивно відсортуємо частини, після чого виконаємо

процедуру злиття: підтримуємо два вказівника, один на поточний елемент першої частини, другий – на поточний елемент другої частини. З цих двох елементів вибираємо мінімальний, вставляємо у результуючий масив і зрушуємо вказівник, що відповідає мінімуму. Злиття працює за O(*n*), рівнів всього *logn*, тому Оцінка числа операцій O(*nlogn*). Ефективно заздалегідь створити тимчасовий масив і передати його в якості аргументу функції. Це сортування рекурсивне, як і *qsort*, а тому можливий перехід на квадратичну асимптотику за невеликого числа елементів.

Сортування підрахунком / *Counting sort*

Створимо додатковий масив частот розміру *r* - l, де l – мінімальний, а *r* – максимальний елемент масиву. Після цього пройдемо за вихідним масивом і підрахуємо кількість входжень кожного елемента, записуючи її у частотний. Тепер можна пройти за масивом частот і виписати кожне число стільки разів, скільки потрібно у результуючий масив. Оцінка числа операцій - O(*n* + *r* - l).

Програма роботи:

1. Обрати за номером варіанту файл (*l6vXX.txt*, де *XX* – номер варіанту) з каталогу вхідних даних для лабораторних робіт №6.
2. Прочитати з файлу масив цілих беззнакових чисел.
3. Обрати за номером варіанту п’ять алгоритмів сортування (див. табл. 6).
4. Реалізувати кожен алгоритм у вигляді функції із такими характеристиками
   a. Приймає масив у вигляді двох аргументів - вказівника та довжини.
   b. Відсортований масив повертає у тій самій пам’яті, де розміщено вихідний.
   c. Приймає також два вказівника на цілі числа, куди записує кількість витрачених операцій та виділеної додаткової пам’яті

d. Повертає ціле число де 0 означає успішне виконання, а -1 – помилку.

e. Ніякі операції вводу-виводу усередині функції сортування не проводити!

5. Створити додатково три масиви цілих чисел такого розміру, як вихідний, один заповнити натуральним рядом чисел у прямому, другий – у зворотному напрямку, третій наполовину нулями, наполовину одиницями.
6. Для кожного алгоритму сортування викликати відповідну функцію для прочитаного з файлу масиву (середній випадок), прямого (кращий) та оберненого (гірший випадок) ряду.
7. Записати виміряні значення кількості операцій для кожного алгоритму та кожного типу масиву вихідних даних у таблицю – має бути записана асимптотика та пам’ять для середнього, кращого, гіршого випадків та випаду з повторами.
8. На підставі презентованої таблиці порівняти обрані алгоритми за критеріями стійкості та природності. Скласти висновки.

Таблиця 6 – Каталог вхідних даних

№ варіанту	Алгоритми сортування				
1	Вставками	Швидке	Вибором	Злиттям	Шейкер
2	Вибором	Швидке	Гномяче	Вставками	Злиттям
3	Шейкер	Вставками	Гребінцем	Швидке	Злиттям
4	Бульбашка	Вставками	Швидке	Злиттям	Гномяче
5	Гребінцем	Злиттям	Вставками	Гномяче	Підрахунком
6	Злиттям	Бульбашка	Вставками	Гребінцем	Підрахунком
7	Гномяче	Бульбашка	Швидке	Вибором	Вставками
8	Швидке	Гномяче	Вибором	Шелла	Вставками
9	Злиттям	Підрахунком	Гребінцем	Гномяче	Швидке
10	Гребінцем	Шелла	Підрахунком	Злиттям	Шейкер
11	Вставками	Гномяче	Злиттям	Вибором	Гребінцем
12	Бульбашка	Злиттям	Гномяче	Швидке	Гребінцем
13	Злиттям	Вставками	Бульбашка	Гномяче	Швидке
14	Бульбашка	Швидке	Гребінцем	Гномяче	Вставками
15	Швидке	Шейкер	Бульбашка	Вставками	Підрахунком
16	Вибором	Вставками	Шейкер	Підрахунком	Шелла
17	Гребінцем	Бульбашка	Гномяче	Підрахунком	Вставками
18	Гномяче	Вставками	Швидке	Гребінцем	Вибором
19	Гребінцем	Гномяче	Підрахунком	Бульбашка	Вставками
20	Злиттям	Вибором	Гномяче	Гребінцем	Підрахунком
21	Бульбашка	Швидке	Шелла	Гребінцем	Підрахунком
22	Гномяче	Бульбашка	Шейкер	Злиттям	Гребінцем
23	Гномяче	Злиттям	Вставками	Вибором	Шелла
24	Шелла	Вставками	Гномяче	Підрахунком	Гребінцем
25	Шелла	Бульбашка	Швидке	Вибором	Гребінцем

Зміст звіту

Звіт слід оформляти в електронному вигляді.

Він повинен містити такі пункти:

1. Мета роботи.

2. Завдання на виконання лабораторної роботи.

3. Лістинг коду програми та результати її виконання.

- Код програми необхідно подати у форматі тексту.
- Результати роботи програми слід навести у вигляді скріншотів або текстового виводу.

4. Висновки.

- Висновки мають бути індивідуальними та відображати сутність виконаної роботи.

Контрольні питання:

1. Які основні характеристики використовуються для оцінки ефективності алгоритмів сортування?
2. У чому різниця між стійкими та нестійкими алгоритмами сортування?
3. Яка асимптотична складність найбільш ефективних алгоритмів сортування?
4. Як працює алгоритм швидкого сортування (*Quicksort*) і в чому його переваги?
5. Який принцип роботи сортування злиттям (*Merge sort*), і в яких випадках воно ефективне?
6. Як порівняти швидкість роботи різних алгоритмів сортування на одному масиві?
7. Які параметри повинні повертати функції сортування у лабораторній роботі?
8. Як змінюється продуктивність алгоритмів у разі роботи з частково відсортованими масивами?
9. Чому алгоритми сортування мають різну продуктивність у кращому, середньому та найгіршому випадках?

10. Які алгоритми сортування найбільш ефективні для великих масивів даних і чому?

Лабораторна робота № 7
Розв’язання систем лінійних рівнянь методом Гауса

Мета роботи: Ознайомитися з методом Гауса для розв’язання систем лінійних рівнянь. Розробити алгоритм та реалізувати програму, яка виконує поетапне приведення системи до верхньої трикутної матриці та знаходить розв’язок методом зворотного ходу. Навчитися працювати з матрицями, виконувати перестановку рядків, контролювати обумовленість системи та перевіряти правильність отриманих розв’язків.

Короткі теоретичні відомості

Метод Гауса (також відомий як метод послідовного виключення невідомих) є одним із фундаментальних та універсальних інструментів у сфері розв’язування систем лінійних алгебраїчних рівнянь (СЛАР) [8]. Цей метод є надзвичайно потужним і використовується у різних наукових та інженерних дисциплінах, де системи лінійних рівнянь зустрічаються на щоденній основі.

Його процес складається з двох ключових етапів: прямого та оберненого ходу.

Прямий хід

На першому етапі, який називається «*прямим ходом*», система лінійних рівнянь піддається елементарним перетворенням з метою зведення до рівносильної системи трикутної форми. Головна мета цього етапу – зробити так, щоб матриця системи стала верхньотрикутною, тобто всі елементи під головною діагоналлю матриці стали рівні нулю.

Елементарні перетворення включають в себе такі операції:

• додавання одного рядка матриці до іншого;

• множення рядка матриці на ненульову константу;

• обмін двох рядків матриці місцями.

Ці операції виконуються, щоб позбавити систему від зайвих невідомих та спростити обчислення.

Обернений хід

Після того як система перетворена у трикутну форму, переходимо до другого етапу – оберненого ходу. На цьому етапі виконується знаходження значень невідомих з отриманої «*ступінчастої*» системи. Починаючи з останнього рядка та обчислюючи значення невідомих за чергою, ми знаходимо розв'язок системи.

Для кращого розуміння методу Гауса і його застосування у розв'язанні систем лінійних алгебраїчних рівнянь, розглянемо процес його використання більш детально.

Перетворення системи у трикутну форму

Почнемо з системи лінійних алгебраїчних рівнянь виду:

$$\begin{cases} a_{11} \cdot x_1 + a_{12} \cdot x_2 + a_{13} \cdot x_3 + \cdots + a_{1n} \cdot x_n = b_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + a_{23} \cdot x_3 + \cdots + a_{2n} \cdot x_n = b_2 \\ \cdots \\ a_{n1} \cdot x_1 + a_{n2} \cdot x_2 + a_{n3} \cdot x_3 + \cdots + a_{nn} \cdot x_n = b_n \end{cases} \quad (1)$$

де a_{ij} – коефіцієнти при x_j у i-му рівняння; b_i – вільний член рівняння.

Припустимо, що коефіцієнт $a_{11} \neq 0$ (цього завжди можна досягнути перестановкою рівнянь). Поділимо коефіцієнти першого рівняння системи на a_{11}, отримаємо:

$$x_1 + c_{12} \cdot x_2 + c_{13} \cdot x_3 + \cdots + c_{1n} \cdot x_n = d_1 \quad (2)$$

де $c_{1j} = a_{1j}/a_{11}$ (для $j > 1$); $d_1 = b_1/a_{11}$.

За допомогою рівняння (2) можна легко виключити невідому x_1 з інших рівнянь системи (1). Це робиться шляхом віднімання від другого рівняння системи (1) рівняння (2), помноженого на a_{21}; від третього рівняння системи (1) віднімається рівняння (2), помножене на a_{31}, і так далі.

Отже, отримаємо систему трикутної форми:

$$\begin{cases} x_1 + c_{12} \cdot x_2 + c_{13} \cdot x_3 + \cdots + c_{1n-1} \cdot x_{n-1} + c_{1n} \cdot x_n = d_1 \\ x_2 + c_{23} \cdot x_3 + \cdots + c_{2n-1} \cdot x_{n-1} + c_{2n} \cdot x_n = d_2 \\ x_3 + \cdots + c_{3n-1} \cdot x_{n-1} + c_{3n} \cdot x_n = d_3 \\ \cdots \\ x_{n-1} + c_{n-1n} \cdot x_n = d_{n-1} \\ x_n = d_n \end{cases}$$

Знаходження значень невідомих

Як вже зазначалося вище, другий етап полягає у знаходженні значень невідомих із системи, яку ми отримали на попередньому етапі. Цей етап називається оберненим ходом, оскільки спочатку, з останнього рівняння, знаходять значення x_n:

$$x_n = d_n = \frac{b_n}{a_{nn}}$$

На наступному кроці, величина x_n підставляється у (n-1)-ше рівняння системи (3), і отже, знаходиться значення x_{n-1}:

$$x_{n-1} = d_{n-1} - c_{n-1\,n} \cdot x_n = \frac{b_{n-1} - a_{n-1\,n} \cdot x_n}{a_{n-1\,n-1}}.$$

Далі, підставивши x_n та x_{n-1} в (n-2)-ге рівняння системи (3), ми знаходимо x_{n-2}. Продовжуючи цей процес далі, ми отримуємо розв’язок системи рівнянь (1).

Очевидно, що даний процес визначений однією формулою:

$$x_k = \frac{1}{a_{kk}} \left(b_k - \sum_{j=k+1}^{n} a_{kj} \cdot x_n \right)$$

Важливі зауваження

- якщо елементи будь-якого рядка матриці системи після перетворень стають рівними нулю, а права частина не дорівнює нулю, то система лінійних рівнянь є несумісною;
- якщо елементи будь-якого рядка матриці системи та права частина після перетворень стають рівними нулю, то система рівнянь є сумісною, але має безліч рішень;
- щоб знайти загальне рішення сумісної системи рівнянь, що має безліч рішень, потрібно у процесі оберненого ходу методу Гауса базисні змінні виразити через вільні змінні;
- базисні змінні у ступінчастій системі – це перші змінні зліва з ненульовим коефіцієнтом або просто перші зліва, тому що змінні з нульовим коефіцієнтом у процесі елементарних перетворень були виключені. Вільні змінні – решта змінних.

Проблеми Розв'язування Систем Рівнянь Використовуючи Метод Гауса: Ділення на Нуль та їх Уникнення

Однією з основних проблем у разі використання методу Гауса для розв'язання систем лінійних рівнянь є можливість ділення на нуль. Ця ситуація виникає, коли ведучий (головний) елемент матриці стає рівним нулю під час прямого ходу методу Гауса.

Наприклад, якщо a_{11} у системі рівнянь стає нулем, то ми не можемо поділити інші коефіцієнти a_{1j} на нуль для знаходження проміжних коефіцієнтів c_{1j}.

Як уникнути ділення на нуль

Для уникнення ділення на нуль існують спеціальні алгоритми та методи:

1. Часткова перестановка: цей метод використовується для вибору ведучого елемента так, щоб уникнути нульових елементів на діагоналі матриці

під час прямого ходу методу Гауса. На кожному кроці вибирається елемент з максимальним за модулем значенням у стовпці, який розташований нижче поточного елемента, і рядки матриці міняються місцями. Це дозволяє уникнути ділення на нуль для виконання наступного кроку.

2. Повна перестановка: у цьому методі на кожному кроці вибирається елемент з максимальним за модулем значенням у підматриці, яка розташована праворуч і нижче поточного елемента. Потім рядки та стовпці матриці міняються місцями, щоб зробити цей елемент діагональним. Це також допомагає уникнути ділення на нуль для виконання наступного кроку.

Вибір методу перестановки

Вибір між частковою та повною перестановкою повинен залежати від конкретної ситуації та характеристик матриці. Повна перестановка може бути більш обчислювально витратною, але дозволяє уникнути ще більше проблем з діленням на нуль. У той час як часткова перестановка може бути більш ефективною у випадках, коли ризик ділення на нуль невеликий.

Програма роботи:

1. Розробити алгоритм та скласти програму, яка розв’язує систему лінійних рівнянь методом Гауса.
2. Програма повинна отримувати від користувача або параметром командного рядка ім'я вхідного файлу і з цього файлу вичитувати параметри системи рівнянь.
3. Дані у вхідному файлі повинні бути представлені у вигляді матриці N x (N + 1) дійсних чисел, розділених пробілами та/або табуляціями. Останній стовпчик матриці представляє вектор вільних членів системи, N – число рівнянь.
4. У межах програми виділити ділянки алгоритму, які вирішують наступні завдання:
   - Читання матриці з файлу з одночасним виділенням під неї пам'яті.
   - Виведення на екран вихідної системи.

- Приведення матриці до діагонального вигляду.
- ***На кожному кроці алгоритму необхідно проводити відбір ведучого елемента (найбільшого за модулем в поточному стовпці нижче головної діагоналі) і здійснювати таку перестановку рядків, щоб він ставав діагональним.***
- ***У процесі розв'язання цього завдання обов'язково контролювати обумовленість системи і, за появи нулів на головній діагоналі, діагностувати розбіжність методу і видати відповідне повідомлення.***
- Знаходження невідомих.
- Перевірка розв'язку підстановкою результату у систему і порівнянням зі стовпчиком вільних членів.
- Виведення на екран результату
- Звільнення раніше виділеної пам'яті

5. У межах програми обов'язково скласти такі незалежні функції:
   - Читання матриці з файлу. (На вході – вказівник на структуру *FILE*, або ім'я файлу і вказівник, що вказує на матрицю; на виході - заповнена матриця та її розміри)
   - Приведення матриці до діагонального вигляду.

Зміст звіту

Звіт слід оформляти в електронному вигляді.

Він повинен містити такі пункти:

1. Мета роботи.

2. Завдання на виконання лабораторної роботи.

3. Лістинг коду програми та результати її виконання.

- Код програми необхідно подати у форматі тексту.
- Результати роботи програми слід навести у вигляді скріншотів або текстового виводу.

3. Висновки.

- Висновки мають бути індивідуальними та відображати сутність виконаної роботи.

Контрольні питання:

1. У чому полягає суть методу Гауса для розв’язання систем лінійних рівнянь?
2. Як відбувається вибір головного елемента (ведучого) у методі Гауса, і чому це важливо?
3. Які основні етапи алгоритму методу Гауса?
4. У чому відмінність між прямим та зворотним ходом методу Гауса?
5. Як визначити, чи має система рівнянь єдиний розв’язок?
6. Що таке погано обумовлена система рівнянь, і як вона впливає на точність розв’язку?
7. Як перевірити правильність знайденого розв’язку після виконання алгоритму?
8. Які переваги та недоліки методу Гауса порівняно з іншими методами розв’язання систем лінійних рівнянь?
9. Як реалізувати вибір ведучого елемента у програмі для підвищення стійкості алгоритму?
10. Як звільнити виділену під матрицю пам’ять після завершення обчислень у програмі?

Лабораторна робота № 8

Маніпуляції зі складними структурами. Нормалізація *WAVE*-файлу

Мета роботи: Ознайомлення з форматом *WAVE*-файлів, вивчення принципів їхньої обробки, а також реалізація алгоритму нормалізації звукових сигналів. Студент має розробити програму, яка зчитує *WAVE*-файл, аналізує його структуру, виконує нормалізацію виборок (збільшення гучності без спотворень) та зберігає результат у новий файл.

Короткі теоретичні відомості

WAVE-формат призначений для зберігання звуку у цифровій формі у вигляді виборок (*samples*). Формат поширений на багатьох платформах. Він побудований на принципі контейнерів (вкладених одна в одну структур). Зовнішній контейнер побудований у форматі *RIFF* [9].

Файл має таку структуру:

- *Заголовок контейнеру RIFF.*
- Тіло контейнеру *RIFF.*
  - Тип даних контейнеру *RIFF* («*WAVE*»).
  - ✓ *Заголовок контейнеру формату.*
  - ✓ Тіло контейнеру формату.
  - ▪ *Заголовок контейнеру даних.*
  - ▪ Тіло контейнеру даних.

Контейнери мають такі структури, див. табл. 7 [10].

Таблиця 7 – Структура контейнера

Зсув	Розмір	Назва за стандартом	Опис	Значення
0x00	4	*Chunk ID*	Назва контейнера	*RIFF*
0x04	4	*Chunk Data Size*	Розмір тіла контейнера	(розмір файла)-8
0x08	4	*RIFF Type*	Тип даних контейнера	*WAVE*
0x0C	4	*Format Chank ID*	Назва контейнера	*fmt*
0x10	4	*Format Chunk Size*	Розмір тіла контейнера	0x10, 0x12, …
0x14	2	*Format tag*	Тип формату	1 – без компресії
0x16	2	*Number of channels*	**Кількість каналів звуку**	1 або 2
0x18	4	*Sample rate*	Швидкість потоку (виборок за секунду)	
0x1*C*	4	*Average bytes per second*	Швидкість потоку (байт за секунду)	
0x20	2	*Block Align*	Вирівнювання виборки	1,2,…
0x22	2	*Bits per sample*	Кількість біт на виборку	8 або 16
0x24	2	*Size of extension*	Розмір додаткового блоку	
0x26	…	*Extension*	Додатковий блок	
…	…	…	…	…
	4	*Data chunk ID*	Назва контейнера	*data*
	4	*Data chunk size*	Розмір тіла контейнера	Сума розмірів усіх виборок
	…	*Samples…*	Виборки	

Слід зауважити, що структура контейнера формату нерегулярна – якщо значення «*Format Chunk Size*» дорівнює 0x10, то поля «*Size of extension*» та «*Extension*» відсутні.

Між контейнерами формату та даних у різних реалізаціях можуть розташовуватись ще кілька контейнерів. Для надійного позиціонування початку контейнера даних, його слід визначати за рядком «*data*».

Виборки у контейнері даних записуються залежно від кількості каналів та кількості біт на виборку.

Для одноканального (моно) звуку, виборки йдуть послідовно, для двоканального (стерео) – за чергою, спочатку виборка першого (лівого) каналу, потім – другого (правого).

Для 8-бітного звуку виборки займають по 1 байту та обробляються, як тип «*unsignrd char*», а за відсутності звуку відповідає значення 128 (0x80). Для 16-бітного звуку виборки займають по 2 байти, та обробляються як signed short і мають значення у діапазоні – 32768 (-0x8000) до 32767 (0x7*fff*), а за відсутності сигналу відповідає значення 0.

Нормалізацією цифрового звуку називають помноження значень виборок на такий коефіцієнт, щоб найбільше за амплітудою значення досягло максимальної для даного типу величини. Отже відбувається підвищення гучності звуку без внесення спотворень.

Програма роботи:

Написати програму, яка виконує такі дії:

1. Отримує з командної строки, або стандартного вводу два імені файлів – вхідного та вихідного.
2. Об’являє *структуру riff_header* згідно наведеної вище таблиці.
3. Об’являє вказівники на масиви значень виборок для чотирьох випадків: 8-біт моно, 8-біт стерео, 16-біт моно, 16-біт стерео.
4. Вичитує з файлу структуру формату.
5. Виходячи зі значень кількості каналів та кількості біт на виборку, виділяє пам’ять під відповідні масиви виборок, обчислює кількість виборок та вичитує виборки у масиви.
6. Знаходить найбільше та найменше значення амплітуди виборки.
7. Обчислює коефіцієнти нормалізації для додатних та від’ємних амплітуд.

8. Виводить у стандартний потік докладні відомості про файл та обчислені коефіцієнти нормалізації.

9. Нормалізує масиви виборок.

10. Записує у вихідний файл структуру формату та масиви виборок.

Записаний файл прослухати за допомогою програвача, та пересвідчитись, що він зібраний правильно.

Зміст звіту

Звіт слід оформляти в електронному вигляді.

Він повинен містити такі пункти:

1. Мета роботи.

2. Завдання на виконання лабораторної роботи.

3. Лістинг коду програми та результати її виконання.

- Код програми необхідно подати у форматі тексту.
- Результати роботи програми слід навести у вигляді скріншотів або текстового виводу.

4. Висновки.

- Висновки мають бути індивідуальними та відображати сутність виконаної роботи.

Контрольні питання:

1. Що таке *WAVE*-формат і для чого він використовується?

2. Яка структура *RIFF*-контейнера у *WAVE*-файлі?

3. Які основні параметри зберігаються у заголовку *WAVE*-файлу?

4. Як визначається кількість каналів у *WAVE*-файлі, і чому це важливо?

5. Що означають параметри *Sample rate*, *Bits per sample*, *Block Align* у форматі *WAVE*?

6. Що таке нормалізація звуку і навіщо вона використовується?

7. Який алгоритм визначення коефіцієнтів нормалізації для додатних та від’ємних амплітуд?

8. Як правильно читати та записувати неформатовані дані з файлу у мові програмування *C*?

9. Які можливі проблеми можуть виникнути у разі обробки стерео- та монофонічного звуку?

10. Як перевірити коректність записаного *WAVE*-файлу після його обробки?

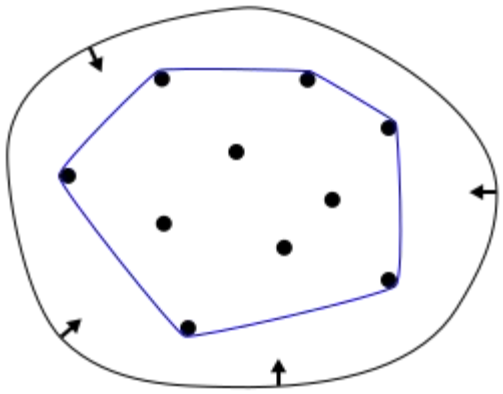
Лабораторна робота № 9

Знаходження опуклої оболонки множини точок на площині. Алгоритм Джарвіса

Мета роботи: Вивчення та реалізація алгоритму Джарвіса (методу загортання подарунка) для знаходження опуклої оболонки множини точок на площині. Студент має розробити програму, яка генерує множину випадкових точок, обчислює їхню опуклу оболонку та визначає, чи знаходиться довільна точка всередині отриманого багатокутника.

Короткі теоретичні відомості

Опуклою оболонкою множини точок називають найменшу опуклу множину (багатокутник), що містить всі точки вихідної множини. Опуклим називають такий багатокутник, що містить у собі відрізок прямої, що з'єднує



будь-які дві його вершини,як показано на рис. 2 [11].

Рисунок 2 – Опукла оболонка множини точок

Для знаходження опуклої оболонки існує чимало алгоритмів – методи Грехема, Джарвіса, Чана, метод еластичної оболонки, *Quickhull* та інші.

Метод Джарвіса (метод загортання подарунка) полягає у послідовному обчисленні полярних кутів між векторами, що задані попередньою знайденою точкою опуклої оболонки, поточною, та рештою множини точок, що не належать до обчисленої частини опуклої оболонки, окрім першої [1]:

$$\vec{a} \quad \vec{b} = |a||b|\sin\theta = \begin{vmatrix} a_x & a_y \\ b_x & b_y \end{vmatrix}.$$

Наступною точкою опуклої оболонки обирається така, що дала найменший полярний кут, як показано на рис. 3 [12].

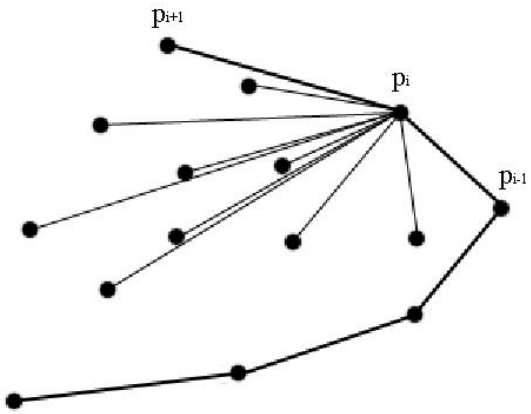


Рисунок 3 – Точки множини опуклої оболонки

За першу точку опуклої оболонки обирають найнижчу найлівішу точку множини. Порівняння полярних кутів на площині зручно зводиться до обчислення псевдоскалярного добутку векторів, що утворені останньою обчисленою точкою опуклої оболонки та двома точками-кандидатами.

Якщо добуток від'ємний – то обирається друга точка. Якщо добуток дорівнює нулю – точки лежать на одній прямій і обирається дальша (з більшим модулем).

Схема алгоритму:

- Знайти найнижчу, найлівішу точку множини *P*, додати її першою до шуканої послідовності, як *p*[0], *i* = 0;
- **do:**

- $p[i+1]$ – будь-яка точка множини, що не входить до шуканої послідовності, але містить першу точку (кандидат у наступні точки опуклої границі);
- для кожної точки $P[j]$ множини, що не входить до шуканої послідовності, але містить першу точку та не містить $p[i+1]$:
  - Знайти псевдоскалярний добуток між векторами ($p[i]$, $p[i+1]$) та ($p[i]$, $P[j]$)
  - Якщо добуток від'ємний – то $p[i+1] = P[j]$
  - Якщо добуток дорівнює нулю - порівняти відстані ($p[i]$, $p[i+1]$) та ($p[i]$, $P[j]$). Якщо друга більша, то $p[i+1] = P[j]$

- i++; **while** $p[i]$! = $p[0]$; – зупинитись, коли наступна обрана точка співпадає з першою.

Отримавши опуклу оболонку множини точок, можна встановити чи знаходиться окрема точка всередині неї. Для цього використовують метод підрахунку «числа обертів» (*winding number*). Метод полягає у тому, що з точки будують промінь, спрямований довільно. Потім для кожного ребра опуклого багатокутника, з'ясовують, чи перетинає воно цей промінь. Якщо перетинає – то перетину присвоюють коефіцієнт +1 або -1 в залежності від того, чи перетин відбувається за годинниковою стрілкою, чи проти. Напрям перетину з'ясовують за знаком скалярного добутку між спрямовуючими векторами променя та ребра у точці перетину. Ознакою того, що точка знаходиться всередині багатокутника, буде ненульова сума отриманих коефіцієнтів.

Програма роботи:

1. Розробити окрему програму, що запитує у користувача максимальні та мінімальні значення за двома координатами, число точок, та генерує множину точок із випадковими значеннями координат у вказаних межах. Точки мають бути представлені у вигляді структур, що складаються з двох реальних змінних *x* та *y*. Отримана множина має бути записана до файлу у бінарному вигляді.
2. Розробити програму, що виконує такі дії:
    a. Отримує у користувача ім'я файлу, що містить множину точок, створену попередньою програмою;
    b. Вичитує з файлу масив структур точок.
    c. Знаходить та друкує межі множини за координатами та координати центроїду множини (середнє значення відповідних координат точок);
    d. Виконує обхід множини за методом Джарвіса та знаходить масив точок, що належать до опуклої оболонки множини. Друкує їх.
3. Написати функцію, яка отримує масив опуклої оболонки та довільну точку, та з'ясовує чи знаходиться точка всередині цієї границі.
4. Протестувати програму та навести у протоколі отримані результати.

Зміст звіту

Звіт слід оформляти в електронному вигляді.

Він повинен містити такі пункти:

1. Мета роботи.

2. Завдання на виконання лабораторної роботи.

3. Лістинг коду програми та результати її виконання.

- Код програми необхідно подати у форматі тексту.
- Результати роботи програми слід навести у вигляді скріншотів або текстового виводу.

4. Висновки.

- Висновки мають бути індивідуальними та відображати сутність виконаної роботи.

Контрольні питання:

1. Що таке опукла оболонка множини точок?
2. Які основні алгоритми використовуються для знаходження опуклої оболонки?
3. У чому полягає суть алгоритму Джарвіса?
4. Як вибирається початкова точка у методі Джарвіса?
5. Яку роль у алгоритмі відіграє обчислення псевдоскалярного добутку?
6. Як перевірити, чи точка належить внутрішній області опуклої оболонки?
7. Що таке «метод підрахунку числа обертів» (*winding number*)?
8. Як можна оптимізувати алгоритм Джарвіса для роботи з великою кількістю точок?
9. Які можливі випадки вирівнювання точок на одній прямій і як їх обробляти?
10. Як згенерувати множину випадкових точок та зберегти її у бінарному файлі?

Лабораторна робота № 10
Стековий парсер

Мета роботи: Вивчення принципів роботи стекових парсерів, розробка алгоритму обробки рядкових виразів та їхнього синтаксичного аналізу. Особливу увагу приділено використанню стекових структур даних для обчислення виразів у зворотній польській нотації та аналізу схем електричних опорів, представлених у вигляді вкладених дужкових записів.

Короткі теоретичні відомості

Аналізатори строкових виразів (парсери) займають важливе місце серед алгоритмів сучасного програмування. Розробка синтаксичних та лексичних аналізаторів вважається окремою дисципліною – комп'ютерною лінгвістикою та стрімко розвивається [13].

Програма, яку ми пишемо мовою Сі, також обробляється досить складними парсерами для верифікації синтаксису та перетворення у зручний для компіляції вигляд.

Основним завданням парсера є як правило перетворення тексту у текст за певними правилами. Процедура розбору спирається на два механізми – лексичний та синтаксичний аналізатори. Завданням лексичного аналізатора є розбиття строки символів на послідовності, які мають синтаксичне значення. Такі послідовності символів називають «токенами» або «лексемами» або просто словами. Послідовність токенів сприймає синтаксичний аналізатор, який спирається на наперед визначену логіку мови і визначає, коли ряд токенів закінчується і прочитане речення потребує певної реакції – виконання якої-небудь дії, або запису певного тексту у вихідний потік.

Синтаксичний аналізатор приймає рішення про завершення синтаксичної конструкції (речення) на підставі останнього токена (термінатора), але з

урахуванням усіх попередніх. Для зберігання попередніх токенів до приходу термінального використовується стек.

Стек – це така послідовна структура даних, у яку можна вміщувати елементи та виймати їх тільки з одного боку. Стек підтримує дві операції – *push*(*x*) – вміщує елемент *x* у вершину стеку, та *pop*(*x*) – повертає черговий елемент з вершини стеку та видаляє його зі стеку.

У даній роботі розглядається функціонал лексичного та синтаксичного аналізу на базі аналізатора виразів. Вирази, наприклад арифметичні, можуть бути подані у інфіксному «*a* + *b*», префіксному «+*ab*» або постфіксному «*ab*+» записі. Останні дві форми називають «польською» та «оберненою польською нотацією» на честь логіка Яна Лукасевича, який у 1920 році запропонував таку форму запису логічних виразів без дужок. Обернений запис зручніший для синтаксичного аналізу і найчастіше використовується у парсерах.

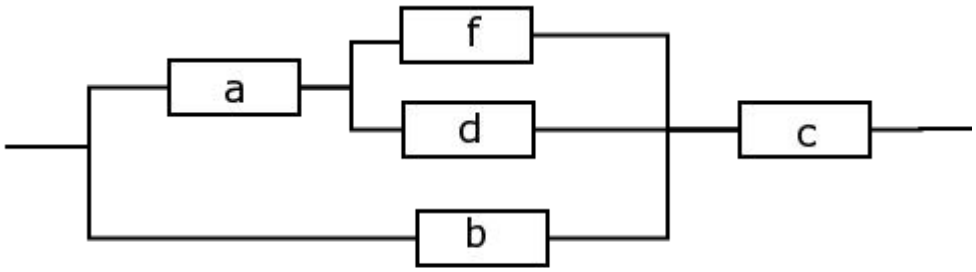
У разі аналізу виразу у оберненій нотації на прикладі наведеного вище, токени, які не є термінаторами, заносяться до стеку аж доки не буде отриманий перший термінатор. У нашому прикладі термінатором є токен "плюс" і він поєднує у синтаксичній конструкції два попередні.

Парсер, отримавши термінатор, виймає зі стеку два попередні токени, проводить із ними потрібну арифметичну операцію і записує результат назад до стеку. Отриманий результат може бути використаний іншими синтаксичними конструкціями, які обгортають дану. Результат виконання всього виразу може бути вийнятий зі стеку після обробки останнього токена. Якщо на той момент у стеку залишилося більше одного елемента – вираз був невалідний.

Завдання:

Резистори у електронних схемах можуть бути поєднані послідовно (тоді $R_{\sum} = R_1 + R_2$) та паралельно (тоді $R_{\sum=} \frac{R_1 \cdot R_2}{(R_1+R_2)}$). Так можна об'єднувати підсхеми, які у свою чергу мають всередині комбінації паралельних та

послідовних з'єднань. Існує форма строкового запису, коли паралельне об'єднання записується у квадратних дужках, а послідовне – у круглих.
Приклад схеми та її запису показаний на рис. 4.



([(a [f d]) b] c)

Рисунок 4 – Схема з’єднання опорів

Такий запис є оберненим, має 5 видів токенів – «відкрита кругла дужка», «закрита кругла», «відкрита квадратна», «закрита квадратна» та «число» і дві синтаксичні конструкції – «послідовне з'єднання» та «параллельне з'єднання». А проте обидві відкриті дужки є надлишковими з точки зору синтаксичного аналізатора і можуть бути проігноровані, за винятком ситуації коли вони використовуються для перевірки синтаксичної валідності запису. Термінаторами обох синтаксичних конструкцій є закриті дужки.
Завданням роботи є:

- Написати парсер, який приймає, наприклад з клавіатури, рядок опису схеми і друкує результат обрахунку сумарного опору, або повідомлення про синтаксичні помилки.
- Числа вважати реальними додатними у форматі *IEEE*754, тобто такими, що містять символи [0-9+-.].
- Розділювачами вважати пробіли або табуляції. Розділювачів може бути кілька поспіль.
- Решту символів вважати синтаксичними помилками.

Структурно програма має містити основну функцію (не main!), яка приймає на вхід строку, а повертає результат, або друкує помилку, функцію

лексичного аналізатора, яка працює у режимі «*GET NEXT*», тобто за кожним викликом повертає наступну лексему рядка аж доки рядок не закінчиться, та функцію синтаксичного аналізатора, яка оперує стеком.

Зміст звіту

Звіт слід оформляти в електронному вигляді.

Він повинен містити такі пункти:

1. Мета роботи.

2. Завдання на виконання лабораторної роботи.

3. Лістинг коду програми та результати її виконання.

- Код програми необхідно подати у форматі тексту.
- Результати роботи програми слід навести у вигляді скріншотів або текстового виводу.

4. Висновки

- Висновки мають бути індивідуальними та відображати сутність виконаної роботи.

Контрольні питання:

1. Що таке парсер і які його основні функції?
2. У чому полягає різниця між лексичним та синтаксичним аналізом?
3. Яка структура та основні операції стеку?
4. Що таке польська та зворотна польська нотація?
5. Як працює стековий парсер для обчислення арифметичних виразів?
6. Як використовувати стек для обробки вкладених дужкових конструкцій?
7. Як визначити синтаксичну коректність виразу зі вкладеними дужками?
8. Які можливі помилки можуть виникнути при парсингу рядкових виразів?
9. Як працює алгоритм обчислення еквівалентного опору електричних схем, представлених у дужковій нотації?
10. Які методи можна використати для оптимізації роботи стекового парсера?

РОЗРАХУНКОВА РОБОТА

Загальні вимоги

Кожна розрахункова робота має бути оформлена у вигляді програми мовою С.

Програма має бути зібрана компілятором *gcc* на навчальному сервері та протестована. Вона має виконуватись у пакетному режимі, тобто не повинна використовувати інтерактивну взаємодію з користувачем (*scanf* тощо).

Якщо у завданні перелічені дані, що подаються «на вхід», це означає, що вони є аргументами командного рядка (*argv*). У окремих випадках, вхідні дані подаються файлом, тоді аргументом командного рядка є шлях до файлу.

Обов'язково зробити перевірку введених даних на відповідність завданню. Вивести діагностику помилок.

Результати роботи програми мають бути надруковані у вихідний потік.

Наприклад, програма *add*, що реалізує додавання двох чисел, має запускатись так:

```
> add  1  2
> 3
```

Індивідуальні завдання до розрахункової роботи наведені у табл.8.

Таблиця 8 – Індивідуальні завдання до розрахункової роботи

№ варіанту	Завдання
1	**Фільтр на напівтонового зображення.** На вхід подають три файли (імена у командному рядку). Перший містить матрицю цілих додатних чисел у діапазоні 0..255 (пікселі зображення). Розмір матриці по стороні не менше 100. Другий містить матрицю реальних чисел (маску) із розмірами по стороні не більше 9. Розміри матриці маски обов'язково мають бути непарними. Сума усіх чисел маски повинна дорівнювати одиниці. Обидві матриці слід вичитати та застосувати наступним чином: На кожний елемент матриці зображення слід накласти маску центральним значенням і знайти суму значень пікселів, що опинились під елементами маски, помножених на відповідні елементи маски. Результат округлити до цілого числа та замінити на нього центральний піксель. Для пікселів, близьких до краю зображення, коли маска вилізає за край, вважати, що віртуальні пікселі за краєм дорівнюють нулю. Результат записати у третій файл. Для підготовки матриці пікселів з реальних зображень можна використати функції Matlab або Octave: % завантажити зображення *a = imread('file.jpg');* % перетворити його на напівтонове *b = rgb2gray(a);* % перетворити напівтонове зображення на матрицю *m = reshape(typecast(b,'uint8'),size(b));* % зберегти матрицю у файл *dlmwrite('filename.txt',m,'delimiter',' ');* % завантажити перероблену матрицю *k = load('filename.txt');* % перетворити матрицю на зображення *p = mat2gray(k);* % показати зображення *imshow(p);*

Продовження таблиці 8

2	**Перетворення систем обчислення.** На вхід передати слово, що містить число у одній з систем обчислення – двійковій, вісімковій, десятковій, або шістнадцятковій. Слід визначити систему за ознаками – двійкове число містить тільки нулі та одиниці і має префікс «0*b*», вісімкове число починається з нуля, шістнадцяткове число має префікс «0*x*». Розпізнавши число, слід вивести його усіма вищеназваними системами числення. Слід врахувати розмір числа у байтах, що має бути 1, 2, 4 або 8 байтів. Числа вважати беззнаковими.
3	**Перетин відрізків.** На вхід передати файл, що містить реальні координати точок у двох колонках x[*i*] *y*[*i*]. Крім імені файлу на вхід передати реальні координати кінців відрізка x_1 y_1 x_2 y_2. Вважаючи, що послідовність точок у файлі є ломаною лінією, знайти всі точки перетину відрізка з командного рядка з відрізками з файлу. Для кожного перетину вивести номер відрізка що має перетин та координати перетину.
4	**Тарифний план.** На вхід передати файл, що містить тарифний план паркування у трьох колонках – час початку інтервалу, час закінчення інтервалу, ціна за годину. Час має бути записаний у вигляді двох цілих чисел через двокрапку. Години подаються у 24г форматі. Рекомендовано зберігати час, переводячи його у реальне значення години з долями годин. Також на вхід передати інтервал паркування, що треба порахувати, у двох значеннях – початок та закінчення. Якщо час закінчення менше часу початку, вважати, що інтервал перетнув північ. Інтервалів довше доби бути не може. Слід порахувати суму, нараховану за паркування, врахувавши, що можуть бути задіяні кілька періодів тарифу одночасно. Якщо певний час не перекритий явно тарифним планом, вважати, що ціна дорівнює нулю.

Продовження таблиці 8

5	**Належність точки до фігури.** На вхід передати файл, що містить дві колонки координат точок $x[i]$ $y[i]$ що є вершинами довільного багатокутника. Також на вхід передати координати точки. Обчислити, чи належить точка багатокутнику.
6	**Генератор гармонійних коливань.** На вхід передати параметри гармонійного сигналу – амплітуду, частоту, постійну складову та початкову фазу. Також на вхід передати інтервал дискретизації та кількість точок. Розрахувати та вивести вказану кількість значень із вказаним шагом для функції сінуса із вказаними параметрами.
7	**Нормалізація тексту.** На вхід передати два файли. Один містить довільний текст, який слід вичитати по рядках. З тексту слід прибрати усі зайві пробіли, додати загублені і виправити капіталізацію. Зайвим вважати пробіл якщо їх більше одного поспіль, або він стоїть після відкриваючою або перед закриваючою дужкою, або розташований на кінці тексту. Загублені пробіли можуть бути після крапки або коми. Помилкою капіталізації вважати речення, що починається з маленької літери після крапки. Виправлений текст записати у другий файл.
8	**Обертання числа.** На вхід передати ціле додатне число у десятковому форматі. Обчислити десяткове число, що отримують з першого, перевернувши розряди навпаки. Наприклад з 12345 отримати 54321. Окремо вказати, якщо число є паліндромом (симетричне).

Продовження таблиці 8

9	**Прості числа.** На вхід передати два цілих числа, друге більше за перше. Знайти всі прості числа у вказаному діапазоні. Рекомендоване використання решета Ератосфена.
10	**Корінь.** На вхід передати число, ступінь та бажану похибку обчислення (10..0.01%). Знайти за вказаної похибки наближене значення кореня вказаного ступеня з числа рекурсивним способом. Вивести значення кореня та число ітерацій.
11	**Генератор прямокутних імпульсів.** На вхід передати параметри прямокутного імпульсу – амплітуду мінімальну, амплітуду максимальну, початкову затримку, довжини переднього та заднього фронту, довжину вершини та період. Також передати інтервал дискретизації та число точок. Розрахувати та вивести вказану кількість значень із вказаним шагом для прямокутного імпульсу (трапеції) із вказаними параметрами.
12	**Детермінант.** На вхід подати файл, що містить квадратну матрицю реальних чисел із розміром по стороні не менше 10. Обчислити детермінант матриці методом декомпозиції рекурсивним способом.
13	**Злиття масивів.** На вхід передати три файли. Перші два містять масиви реальних чисел, наперед відсортовані. У третій файл записати результат злиття перших двох масивів із збереженням сортування. Злитий масив не пересортовувати.

Продовження таблиці 8

14	**Створення словника.** На вхід передати два файли. Першій містить довільний текст. Текст вичитати, розбити на слова, прибрати усі пробіли та знаки пунктуації, перевести в нижній регістр та відсортувати слова за алфавітом, прибравши дублікати. Список слів вивести у другий файл.
15	**Перевірка тесту.** На вхід подати файл з результатами тесту з арифметики. Перша та третя колонки – реальні числа, друга колонка – знак операції [+ - * /], четверта - результат. Перевірити чи вірний результат кожної операції. Вивести відсоток вірних результатів, привести його до 100-бальної шкали та поставити оцінку ECTS.
16	**Інтервал.** На вхід подати дві дати з часом у форматі DD-MM-YYYYThh:mm:ss. Обидві дати нашої ери. Розрахувати та вивести інтервал між точками у секундах.
17	**Досконалі числа.** Натуральне число називається досконалим, якщо воно дорівнює сумі всіх своїх дільників крім самого себе. Наприклад, 6=1+2+3. На вхід дано натуральне число. Отримати всі досконалі числа, менші за нього.
18	**Подібність масивів.** На вхід передати два файли, що містять реальні масиви. Якщо масиви різної довжини – вкоротити довгий до короткого. Поділити масиви один на другий поелементно, для відношень знайти середнє значення, нормалізувати масив відношень на нього та знайти середньоквадратичне відхилення нормованого масиву. Цю величину вивести як міру відстані між масивами.

Продовження таблиці 8

19	**Календар.** На вхід подати три цілих числа, що містять день, місяць та рік (нашої ери). Обчислити день тижня.
20	**Парабола.** Подати на вхід реальні числа A, B, C, D та координати точки x, y. Визначити, чи належить точка (x,y) області, що обмежена параболою $y = Ax^2+B$ та прямою $y = Cx + D$
21	**Римські цифри для реальних чисел.** На вхід подати слово, що містить додатне реальне число менше 4000 у десятковій формі (арабськими цифрами), або у римській формі (латинськими літерами). Врахувати правило римського запису дробної частини числа у дванадцятковій формі за допомогою точок та літери S (semis). Дробну частину десяткового числа обмежити 4 розрядами. Наприклад: 128.5833 = CXXVIIIS. Розібрати введене число і вивести його у обох формах. Розрахувати та вивести похибку округлення для обох перетворень.
22	**Сортування дат.** На вхід подати два файли. Перший містить масив дат у форматі DD-MM-YYYY. Відсортувати масив за зростанням та вивести у другий файл.
23	**Характеристика матриці.** На вхід подати файл, що містить реальну матрицю розміром по стороні від 7 до 20. Обчислити першу, другу та третю норми матриці та числа обумовленості для цих трьох норм.

Продовження таблиці 8

24	**Перестановка цифр.** На вхід подати слово що містить ціле додатне число у десятковій формі з числом цифр від 2 до 10. Обчислити найбільше і найменше число, яке можна записати цифрами введеного числа так, щоб воно вклалось у тип unsigned int. Вивести результат як цілі числа.
25	**Згладжування масиву.** На вхід подати ціле число ітерацій та два файли. Перший файл містить реальний масив. Кожен елемент масиву замінити середнім арифметичним із його сусідами. Процедуру повторити стільки разів, скільки вказано у числі ітерацій. Результат записати у другий файл.
26	**Пошук двомірного локального мінімуму.** На вхід подати файл, що містить матрицю реальних чисел розміром по стороні не менше 20. У матриці знайти локальний мінімум спрощеним методом градієнтного спуску наступним чином: - Обрати довільну точку початку (x,y) - Для всіх 8 напрямків руху до сусідніх клітинок матриці обчислити швидкість спуску, як різницю значень у поточній та сусідній клітинках, поділену на довжину шагу. - Довжиною шагу вважати одиницю для шагу вздовж клітинки, або корінь з двох для шагу по діагоналі. - Шаги, що ведуть за межі матриці проігнорувати. - З усіх шагів обрати той, де швидкість спуску найбільша та перемістити туди точку. - Якщо усі шаги дають нульову швидкість – робити шаг у напрямку (+1 +1), якщо такий шаг веде за межі матриці – завершити висновком, що мінімум не знайдений. - Якщо всі швидкості від’ємні – закінчити висновком, що мінімум знайдений та вивести його координати та значення.

Продовження таблиці 8

27	**Факторіали.** На вхід передати ціле додатне число *n* та цілу додатну кратність *k* від 1 до 9. Обчислити та вивести для числа *n* всі факторіали кратності від 1 до *k*. Застосувати рекурсивні функції. *n*!, *n*!!, *n*!!!, ….
28	**Концентрації.** На вхід подати бажану концентрацію діючої речовини, як реальне число менше 1 та файл, що містить у двох колонках кількість розчину в літрах та концентрацію. Розрахувати кількість розчину та концентрацію, що утвориться при змішуванні усіх розчинів, зазначених у файлі. Розрахувати, скільки діючої речовини чи навпаки – води, слід додати до розчину, щоб отримати бажану концентрацію.
29	**Раціональні дробі.** На вхід подати додатне реальне десяткове число. Якщо десяткове число є періодичним дробом – періодична частина береться у дужки. Обчислити його форму у вигляді цілої частини та звичайної дробі. Наприклад: 0.75 = 3/4, 1.88 = 1 22/25, 0.(7) = 7/9
30	**Зсув часу.** На вхід подати момент часу у форматі DD-MM-YYYYThh:mm:ss та реальне число годин зсуву. Розрахувати момент часу, зсунутого на вказану величину та надрукувати у такому ж форматі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Бородкіна І.Л., Бородкін Г.О. Теорія алгоритмів: Посібник для студентів вищих навчальних закладів. – К.: Національний університет біоресурсів та природокористування України, 2018. – 231 с.
2. Захаров В.В., Мирончук О. А. Основи програмування: навчальний посібник. – Київ: Ліра-К, 2020.
3. Мельник А. О. Основи програмування та алгоритмізації: навч. посібник. – Тернопіль: ТНТУ, 2021.
4. Львов М.С., Співаковський О.В. Основи алгоритмізації та програмування: Навч. посібник. – Херсон, 1997. – 371 с.
5. Ткачук В.М. Алгоритми і структури даних: Навчальний посібник. – Івано-Франківськ: Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016. – 286 с.
6. Грицюк Ю.І., Рак Т.Є. Об'єктно-орієнтоване програмування мовою С++: навчальний посібник. – Львів : Вид-во Львівського ДУ БЖД, 2011. – 404 с.
7. Грицюк Ю.І., Рак Т.Є. Програмування мовою С++ : навчальний посібник. – Львів : Вид-во Львівського ДУ БЖД, 2011. – 292 с.
8. Вінник В.Ю. Алгоритмічні мови та основи програмування: мова С / В.Ю. Вінник - Житомір :ЖДТУ,2007. – 328 с.
9. Креневич А. П. С у задачах і прикладах : навчальний посібник із дисципліни "Інформатика та програмування" / А.П. Креневич, О.В. Обвінцев. – К. : Видавничо-поліграфічний центр "Київський університет", 2011. – 208 с.
10. Креневич А. П. С у задачах і прикладах : навч. посібник / А. П. Креневич, О. В. Обвінцев. – Київ : ВПЦ “Київський університет”, 2012. – 212 с.
11. Кормен Т. Х., Лейзерсон Ч. Е., Рівест Р. Л., Штайн К. Алгоритми: побудова та аналіз. – Київ: Видавництво «Вільямс», 2020.
12. Дьяків О. І., Качмар В. М. Основи алгоритмізації та програмування. – Львів: Видавництво Львівської політехніки, 2019.
13. Стеценко В. В. Основи програмування мовою С. – Київ: Каравела, 2018.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. В.В. Бублик, В.В. Личман, О.В. Обвінцев. Конспект лекцій з курсу "Інформатика та програмування" [Електронний ресурс] –Режим доступу до ресурсу: http://matfiz.univ.kiev.ua/informatics/lectures/

2. Бєлов Ю.А., Карнаух Т.О., Коваль Ю.В., Ставровський А.Б. 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.

3. Трофименко О.Г., Прокоп Ю.В., Швайко І.Г. та ін.С++. Основи програмування. Теорія та практика:Підручник / О.Г. Трофименко, Ю.В. Прокоп, І.Г. Швайко, Л.М. Буката, Л.А. Косирева, Ю.Г. Леонов, В.В. Ясинський; за ред. О.Г. Трофименко. — Одеса: Фенікс, 2010. — 544 с.

4. Електронний довідник бібліотек С++ [Електронний ресурс] –Режим доступу до ресурсу: http://www.cplusplus.com/reference/clibrary/

5. Електронний підручник з С++[Електронний ресурс] –Режим доступу до ресурсу: https://www.learncpp.com/

6. Відкритий навчальний посібник C Tutorial [Електронний ресурс] –Режим доступу до ресурсу: https://www.tutorialspoint.com/cprogramming/index.htm

7. Електронний довідник C language [Електронний ресурс] –Режим доступу до ресурсу: https://en.cppreference.com/w/c/language

8. Відкритий навчальний посібник C Programming [Електронний ресурс] Режим доступу до ресурсу: https://www.eskimo.com/~scs/cclass

9. ANSI 89 –Programming Language C: American National Standard for Information Systems : ANSI X3.159-1989 - 338c.

10. Стандарт C99 [Електронний ресурс] –Режим доступу до ресурсу: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf

11. Стандарт C11 [Електронний ресурс] –Режим доступу до ресурсу: http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf

12. Стандарт C17 [Електронний ресурс] –Режим доступу до ресурсу: https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2347.pdf

13. Стандарт C23 [Електронний ресурс] –Режим доступу до ресурсу: https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3220.pdf

14. CERT C Coding Standard [Електронний ресурс] –Режим доступу до ресурсу:
https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard

15. Стандарт для роботи с широкими символами [Електронний ресурс] – Режим доступу до ресурсу:

16. Відкритий підручник по C++ українською мовою[Електронний ресурс] Режим доступу до ресурсу: https://purecodecpp.com/uk/archives/920

17. Вакал Є.С.,Личман В.В.,Обвінцев О.В.,Бублик В.В.,Попов В.В. Задачі до курсу «Інформатика та програмування». [Електронний ресурс] –Режим доступу до ресурсу:

18. Збірник задач з дисципліни "Інформатика і програмування" Вакал Є.С., Личман В.В., Обвінцев О.В., Бублик В.В., Довгий Б.П., Попов В.В. -2-ге видання, виправлене та доповнене –К.:ВПЦ "Київський університет", 2006.–94с.

19. Zelle, J. M. Python Programming: An Introduction to Computer Science. — Franklin, Beedle & Associates Inc., 2017.

20. Knuth, D. E. The Art of Computer Programming. Vol. 1–3. — Addison-Wesley, 2011.

21. Forouzan B. A. Foundations of Computer Science. — Cengage Learning, 2007.

22. King K. N. C Programming: A Modern Approach. — W. W. Norton & Company, 2008.

23. Downey A. B. Think Python: How to Think Like a Computer Scientist. — O’Reilly Media, 2015.

24. Hunt A., Thomas D. The Pragmatic Programmer: Your Journey to Mastery. — Addison-Wesley, 2019.

25. Sedgewick R., Wayne K. Algorithms (4th Edition). — Addison-Wesley, 2011.

26. Weiss M. A. Data Structures and Algorithm Analysis in C. — Pearson, 2013.

27. Levitin AV (2007) Introduction to the Design and Analysis of Algorithms, 2-nd Edition. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA
28. Deitel, Harvey M. and Deitel, Paul J.. C how to program.. : Prentice Hall, 2009.
29. Brian W. Kernighan and Dennis M. Ritchie. 1988. The C Programming Language (2nd. ed.). Prentice Hall Professional Technical Reference. ISBN: 0131103709
30. Aho, A. V., Hopcroft, J. E.,, Ullman, J. D. (1983). Data Structures and Algorithms. Reading, Massachusetts: Addison-Wesley. ISBN: 0201000237
31. IEEE Standard for Binary Floating-Point Arithmetic. IEEE Std 754-2008 (Revision of IEEE Std 754-1985). — NY: IEEE, 2008. — 70 p.
32. Wood D (1993) Data Structures, Algorithms, and Performance. Addison-Wesley
33. Publishing Co., Reading, MA, USA
34. Hopcroft JE, Motwani R, Ullman JD (2001) Introduction to automata theory, languages, and computation, 2nd edition. SIGACT News 32(1):60–65, DOI http://doi.acm.org/10.1145/568438.568455