МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кафедра комп'ютерної інженерії

Дисципліна: Програмування
Спеціальність: 123 «Комп’ютерна інженерія»
Освітній рівень: Перший (бакалаврський)
Семестр: І

ЛАБОРАТОРНА РОБОТА № 3

Тема: Одновимірні та багатовимірні масиви, обробка символьних рядків у мові С та робота з гілками в Git

---

1. МЕТА РОБОТИ

- Набути практичних навичок обробки послідовних областей оперативної пам'яті: проектування алгоритмів фільтрації, пошуку екстремумів та перестановок елементів в одновимірних масивах.
- Опанувати кеш-дружні алгоритми обробки двовимірних матриць з урахуванням апаратного стандарту розташування даних за рядками (Row-Major Order).
- Засвоїти низькорівневу специфіку роботи з текстовою інформацією через символьні масиви типу `char` із кінцевим нуль-термінатором (`\0`).
- Опанувати командну модель розгалуження в системі контролю версій Git: створення функціональних гілок (`git branch` , `git checkout` / `git switch`), фіксація змін та їхнє безпечне злиття в основну гілку (`git merge`).
- Підготувати підсумкову інженерну документацію та долучити **обов'язкову пояснювальну записку у форматі PDF** до звітності.

---

2. НЕОБХІДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

1. Текстовий редактор / середовище розробки: **Visual Studio Code** (із розширенням *C/C++* від Microsoft) або **CLion**.
2. Компілятор: **GCC** (GNU Compiler Collection).

3. Система контролю версій: **Git**.
4. Програма для підготовки інженерної документації та експорту в PDF (**LaTeX**, **MS Word** або **LibreOffice Writer**).

---

3. ПОСЛІДОВНІСТЬ ВИКОНАННЯ РОБОТИ

Етап 0. Робота з гілками в Git (Branching Workflow)

1. Відкрийте термінал і перейдіть до кореневого каталогу вашого репозиторію:
   ```
   cd ~/projects/programming-sem1
   ```
2. Переконайтеся, що ви перебуваєте в основній гілці `main`, та створіть нову робочу гілку `lab03`:
   ```
   git checkout -b lab03
   # або альтернативна команда: git switch -c lab03
   ```
3. Створіть каталог для поточної лабораторної роботи та перейдіть у нього:
   ```
   mkdir lab03
   cd lab03
   ```

Етап 1. Обробка одновимірного масиву (`task1.c`)

1. У каталозі `lab03` створіть файл `task1.c`.
2. Відповідно до свого номера варіанта з **Таблиці 1** реалізуйте програму, яка:
   - Оголошує статичний масив фіксованого розміру через константу препроцесора (наприклад, `#define SIZE 15`).
   - Заповнює масив псевдовипадковими цілими числами з діапазону $[-50; 50]$ за допомогою функцій `rand()` та `srand()` із бібліотек `<stdlib.h>` та `<time.h>`.
   - Виводить вихідний масив у консоль в один рядок із чітким форматуванням.
   - Виконує індивідуальне обчислення відповідно до варіанта (пошук екстремумів, сум, добутків або перестановку елементів). При пошуку максимуму чи мінімуму обов'язково ініціалізуйте еталонну змінну значенням нульового елемента масиву!
   - Виводить отримані результати та стан перетвореного масиву.
3. Скомпілюйте програму з прапорцем виявлення всіх попереджень:
   ```
   gcc -Wall -Wextra task1.c -o task1
   ./task1
   ```

4. Зафіксуйте створений сирцевий файл у робочій гілці Git:

```
git add task1.c
git commit -m "feat(lab3): implement 1D array operations"
```

Етап 2. Кеш-дружня обробка 2D-матриць (`task2.c`)

1. У каталозі `lab03` створіть файл `task2.c` .
2. Відповідно до свого номера варіанта з **Таблиці 2** реалізуйте програму для роботи з прямокутною або квадратною матрицею розміром $R \times C$ (наприклад, 4×4 або 3×5):
   - Заповніть матрицю цілими чи дійсними числами.
   - **Обов'язкова вимога швидкодії:** Організуйте вкладені цикли за принципом **Row-Major Order** (зовнішній цикл перебирає індекс рядків, внутрішній — індекс стовпчиків), що гарантує лінійне проходження пам'яті та ефективну роботу кеш-ліній процесора.
   - Виконайте індивідуальне матричне перетворення (операції з діагоналями, пошук сідлових точок, перестановки колонок/рядків).
   - Виведіть матрицю до та після обробки у вигляді акуратної таблиці з фіксованою шириною полів (`%6d` або `%8.2lf`).
3. Скомпілюйте та зафіксуйте зміни в Git:

```
gcc -Wall -Wextra task2.c -o task2
git add task2.c
git commit -m "feat(lab3): implement cache-friendly 2D matrix
transformation"
```

Етап 3. Ручна низькорівнева обробка C-рядків (`task3.c`)

1. У каталозі `lab03` створіть файл `task3.c` .
2. Відповідно до свого номера варіанта з **Таблиці 3** реалізуйте обробку тексту:
   - Виділіть буфер символів із запасом пам'яті під нуль-термінатор згідно з правилом « $N + 1$ байт».
   - Для зчитування рядка використовуйте безпечну функцію `fgets(str, sizeof(str), stdin)` , яка запобігає переповненню буфера на стеку.
   - **Головна інженерна вимога:** Алгоритм аналізу та модифікації тексту має бути реалізований **вручну через покрокове сканування масиву в циклі до байта** `\0` . Заборонено використовувати готові модифікуючі функції бібліотеки `<string.h>` !

- Переконайтеся, що після модифікацій у кінці тексту коректно збережено завершальний нуль-термінатор.

3. Скомпілюйте та зафіксуйте код у Git:

```
gcc -Wall -Wextra task3.c -o task3
git add task3.c
git commit -m "feat(lab3): implement manual null-terminated string parsing"
```

Етап 4. Долучення звіту, злиття гілок та відправка на GitHub

1. Підготуйте технічний звіт та збережіть його **виключно у форматі PDF** із точною назвою `Report_Lab03.pdf` у каталозі `lab03`.
2. Зафіксуйте файл звіту в робочій гілці:

```
git add Report_Lab03.pdf
git commit -m "docs(lab3): add mandatory engineering PDF report"
```

3. Перемкніться на основну гілку `main` та виконайте злиття напрацювань:

```
git checkout main
# або: git switch main
git merge lab03
```

4. Надішліть усі оновлення обох гілок на віддалений сервер GitHub:

```
git push origin main
git push origin lab03
```

---

4. ІНДИВІДУАЛЬНІ ВАРІАНТИ ЗАВДАНЬ

Номер варіанта визначається номером у списку академічного журналу групи:

$$\text{Варіант} = (\text{Номер у списку} - 1) \pmod{15} + 1$$

Таблиця 1. Варіанти до Завдання 1 (Одновимірні масиви)

Вар.	Індивідуальне розрахункове завдання
1	Знайти мінімальний за модулем елемент масиву та суму елементів, розташованих після першого від'ємного.

Вар.	Індивідуальне розрахункове завдання
2	Знайти максимальний елемент масиву та добуток елементів, розташованих між першим і останнім нульовими.
3	Знайти номер мінімального елемента та суму елементів масиву, розташованих між першим і останнім додатними.
4	Порахувати кількість елементів у діапазоні $[A, B]$ та виконати реверс елементів, розташованих після максимального.
5	Знайти суму додатних елементів масиву та змінити знак усіх елементів, розташованих до першого мінімального.
6	Знайти добуток елементів із парними індексами та поміняти місцями мінімальний і максимальний елементи.
7	Визначити кількість від'ємних елементів та замінити всі парні за значенням числа нулями.
8	Знайти суму елементів масиву, розташованих після першого максимального елемента.
9	Здійснити циклічний зсув усіх елементів масиву праворуч на одну позицію (останній стає нульовим).
10	Знайти середнє арифметичне додатних чисел і поділити на нього всі від'ємні елементи масиву.
11	Виконати повний реверс усього масиву «на місці» (In-Place Reversal) без використання допоміжних масивів.
12	Знайти другий за величиною елемент масиву (другий максимум) за один лінійний прохід циклу.
13	Стиснути масив, видаливши з нього всі нулі (зсув ненульових елементів ліворуч із заповненням кінця нулями).
14	Порахувати кількість інверсій у масиві (пар індексів $i < j$, для яких виконується $A[i] > A[j]$).
15	Знайти довжину найдовшої неперервної послідовності однакових чисел, що йдуть підряд.

Таблиця 2. Варіанти до Завдання 2 (Двовимірні матриці)

Вар.	Індивідуальне матричне завдання
1	Для кожної колонки матриці знайти мінімальний елемент та поміняти місцями перший і останній рядки.

Вар.	Індивідуальне матричне завдання
2	У квадратній матриці обчислити суму елементів, розташованих нижче головної діагоналі, та обнулити побічну діагональ.
3	Знайти рядок із максимальною сумою елементів та інвертувати знаки всіх його елементів на протилежні.
4	Виконати транспонування квадратної матриці «на місці» (дзеркальний обмін елементів відносно головної діагоналі).
5	Порахувати кількість додатних чисел у кожному рядку та зберегти їх в окремий масив-вектор.
6	Знайти координати сідлових точок матриці (елементів, які одночасно є мінімальними у рядку і максимальними у колонці).
7	Перевірити, чи є квадратна матриця симетричною відносно своєї головної діагоналі ($A[i][j] == A[j][i]$).
8	Впорядкувати рядки матриці за зростанням суми їхніх елементів.
9	Замінити нулями всі елементи матриці, які за модулем менші за середнє арифметичне всіх її значень.
10	Поміняти місцями стовпчик із максимальним елементом матриці та стовпчик із мінімальним елементом.
11	У квадратній матриці знайти максимальний елемент серед тих, що лежать одночасно на головній і побічній діагоналях.
12	Здійснити циклічний зсув рядків матриці вниз на одну позицію (останній рядок стає першим).
13	Обчислити добуток усіх ненульових елементів для кожного стовпчика матриці окремо.
14	Дзеркально відобразити елементи прямокутної матриці відносно її вертикальної осі симетрії.
15	Перетворити квадратну матрицю на верхньотрикутну (всі елементи строго під головною діагоналлю замінити нулями).

Таблиця 3. Варіанти до Завдання 3 (Символьні рядки)

Вар.	Індивідуальне завдання обробки тексту
1	Підрахувати загальну кількість слів у введеному тексті (слова розділені одним або кількома пробілами).

Вар.	Індивідуальне завдання обробки тексту
2	Перевірити, чи є введений рядок паліндромом (читається однаково зліва направо і навпаки без урахування пробілів).
3	Перетворити всі малі латинські літери рядка на великі шляхом прямого зсуву кодів таблиці ASCII.
4	Видалити з рядка всі цифрові символи, зсунувши залишок тексту ліворуч зі збереженням кінцевого нуль-термінатора.
5	Замінити всі послідовності з кількох пробілів, що йдуть підряд, на один одиночний пробіл.
6	Знайти довжину найдовшого слова у введеному реченні та вивести це слово в консоль.
7	Перевернути кожне окреме слово в реченні задом наперед, зберігши початковий порядок слів та пробілів.
8	Підрахувати частоту входження кожного голосного символу латинського алфавіту в тексті.
9	Видалити з рядка всі символи пунктуації (крапки, коми, тире, знаки оклику та питання).
10	Продублювати кожну велику літеру в тексті (врахувати запас пам'яті приймального буфера!).
11	Визначити, чи містить введений рядок символи, які повторюються три або більше разів поспіль.
12	Знайти перше слово в реченні, яке починається і закінчується на одну й ту саму літеру.
13	Замінити всі входження символу табуляції `\t` на три звичайні символи пробілу.
14	Видалити з введеного тексту перше і останнє слово речення.
15	Порахувати окремо кількість літер, кількість цифр та кількість розділових знаків у введеному буфері.

---

5. ВИМОГИ ДО ЗВІТНОСТІ ТА ЗДАЧІ РОБОТИ

Робота захищається очно в комп'ютерному класі.
Обов'язковою умовою допуску до захисту є наявність у репозиторії на GitHub **пояснювальної записки у форматі PDF** (`Report_Lab03.pdf`), яка містить: титульний аркуш, формулювання варіанта, лістинги програм `task1.c` , `task2.c` , `task3.c` , скріншоти тестування та висновки.

Структура репозиторію на GitHub перед захистом повинна мати такий вигляд:

```
programming-sem1/
├── .gitignore
├── README.md
├── lab01/
├── lab02/
└── lab03/
    ├── task1.c
    ├── task2.c
    ├── task3.c
    └── Report_Lab03.pdf
```

Примітка щодо Git: Викладач перевіряє наявність окремої гілки `lab03` та історію її злиття з гілкою `main` через візуалізатор мережі на GitHub або команду `git log --graph --oneline`.

---

6. КОНТРОЛЬНІ ЗАПИТАННЯ ДЛЯ ОЧНОГО ЗАХИСТУ

1. За якою формулою процесор обчислює фізичну адресу елемента двовимірної матриці $A[i][j]$ в оперативній пам'яті? Чому кількість колонок є обов'язковим параметром цієї формули?
2. Що таке стандарт Row-Major Order? Чому постовпчиковий обхід великої матриці спричиняє явище Cache Thrashing і сповільнює роботу процесора у десятки разів?
3. Чому для надійного збереження тексту довжиною 15 символів буфер у пам'яті зобов'язаний мати розмір щонайменше 16 байтів?
4. Що станеться під час виклику функції друку тексту, якщо в символьному масиві буде пошкоджено або видалено нуль-термінатор (`\0`)?
5. Чим фізично відрізняється рядок у сегменті пам'яті `.rodata` від символьного масиву на стеку? Що станеться, якщо спробувати модифікувати рядковий літерал?
6. Які команди Git використовуються для створення нової гілки, перемикання між ними та злиття змін у гілку `main`?
7. Що таке покажчик `HEAD` у системі контролю версій Git і на що він вказує в момент створення нової гілки?
8. У чому полягає алгоритмічна перевага реверсу масиву «на місці» (In-Place Reversal) порівняно зі створенням допоміжного масиву-копії?