МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кафедра комп'ютерної інженерії

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

ЛАБОРАТОРНА РОБОТА № 5

Тема: Пряма адресація та арифметика покажчиків, емуляція апаратних регістрів мікроконтролера, передача параметрів за адресою та вирішення конфліктів злиття (Merge Conflicts) у Git

---

1. МЕТА РОБОТИ

- На практиці опанувати пряму роботу з оперативною пам'яттю комп'ютера за допомогою покажчиків та реалізувати механізм передачі параметрів за адресою (Pass-by-Address).
- Опанувати швидкісну адресну навігацію по суцільних масивах даних за допомогою арифметики покажчиків (`ptr++` , `ptr + i` , різниця покажчиків `ptrdiff_t`) без застосування оператора індексації у квадратних дужках.
- Дослідити модифікацію апаратних регістрів «на місці» через покажчики з використанням чотирьох канонів бітових масок (встановлення, скидання, інвертування, тестування).
- Засвоїти інженерні стандарти захисту пам'яті за допомогою кваліфікатора `const` та перевірок на нульовий покажчик (`NULL`).
- Опанувати ключову навичку командної розробки в Git: навмисне створення, аналіз маркерів колізій та **ручне вирішення конфлікту злиття (Merge Conflict)**.
- 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 технічної документації та експорту в PDF (**LaTeX**, **MS Word** або **LibreOffice Writer**).

---

3. ПОСЛІДОВНІСТЬ ВИКОНАННЯ РОБОТИ

Етап 0. Симуляція та ручне вирішення Git Merge Conflict (Обов'язково!)

1. Відкрийте термінал, перейдіть до репозиторію та переконайтеся, що ви перебуваєте в гілці `main`:
```
cd ~/projects/programming-sem1
git checkout main
```
2. Створіть нову робочу гілку `lab05` та перемкніться на неї:
```
git checkout -b lab05
# або: git switch -c lab05
```
3. У корені репозиторію створіть текстовий файл `conflict_test.txt` із таким вмістом:
```
Версія налаштувань контролера: Базова конфігурація периферії
```
4. Зафіксуйте файл у гілці `lab05`:
```
git add conflict_test.txt
git commit -m "feat(lab5): add base peripheral config in lab05"
```
5. Поверніться в гілку `main` і змініть цей самий рядок у файлі `conflict_test.txt` на інший текст:
```
git checkout main
# Змініть вміст conflict_test.txt на: "Версія налаштувань контролера:
Пріоритетна конфігурація MAIN"
git add conflict_test.txt
git commit -m "feat: update peripheral config in main branch"
```
6. Спробуйте виконати злиття гілки `lab05` в гілку `main`:

```
git merge lab05
```

Результат: Система Git заблокує автоматичне злиття та повідомить: `CONFLICT (content): Merge conflict in conflict_test.txt. Automatic merge failed`.

7. **Ручне усунення конфлікту:** Відкрийте файл `conflict_test.txt` у редакторі, знайдіть конфліктні маркери:

```
<<<<<<< HEAD
Версія налаштувань контролера: Пріоритетна конфігурація MAIN
=======
Версія налаштувань контролера: Базова конфігурація периферії
>>>>>>> lab05
```

Видаліть службові рядки `<<<<<<<`, `=======`, `>>>>>>>` вручну, сформуйте єдиний компромісний інженерний текст (наприклад: `Версія налаштувань контролера: Узгоджена фінальна конфігурація`), збережіть файл і завершіть злиття комітом:

```
git add conflict_test.txt
git commit -m "fix(git): resolve merge conflict in controller configuration"
```

8. Створіть робочу теку `lab05` для поточної роботи:

```
mkdir lab05
cd lab05
```

Етап 1. Вихідні параметри та передача за адресою (`task1.c`)

1. У каталозі `lab05` створіть файл `task1.c`.
2. Відповідно до свого номера варіанта з **Таблиці 1** реалізуйте підпрограму комплексного аналізу вхідних даних:
   - Функція повинна приймати вхідні параметри за адресою через покажчики з кваліфікатором `const` (захист вхідних даних від модифікації).
   - Функція **повинна повертати статус завершення** через інструкцію `return` (код 0 — успішне обчислення, код 1 — помилка передачі покажчика `NULL` або некоректні вхідні межі).
   - Розраховані вихідні параметри (мінімум два значення за варіантом) функція зобов'язана повертати **через вихідні параметри-покажчики (Out-Parameters)**, розіменовуючи їх ліворуч від знака присвоєння.
   - У функції `main` оголосіть локальні змінні для результатів, передайте їхні адреси через амперсанд (`&`) у підпрограму та виведіть отримані результати без використання глобальних змінних.

3. Скомпілюйте програму з повним контролем попереджень:

```
gcc -Wall -Wextra task1.c -o task1
./task1
```

4. Зафіксуйте код у Git:

```
git add task1.c
git commit -m "feat(lab5): implement pass-by-address with multiple out-parameters"
```

Етап 2. Модифікація 32-бітного регістра «на місці» через покажчик (`task2.c`)

1. У каталозі `lab05` створіть файл `task2.c` .
2. Відповідно до свого номера варіанта з **Таблиці 2** реалізуйте систему керування апаратним 32-бітним регістром периферійного пристрою (тип `uint32_t` із `<stdint.h>`):
   - Напишіть функцію конфігурації, яка приймає покажчик на цільовий регістр: `void configure_register(uint32_t *reg)` .
   - Реалізуйте захист: якщо `reg == NULL` — функція негайно перериває роботу.
   - За допомогою оператора розіменування та побітових операцій з масками функція повинна **прямо в пам'яті за цією адресою**:
     1. Встановити цільовий біт конфігурації в `1` через операцію «АБО».
     2. Скинути біт блокування в `0` через операцію «І» з інверсією маски.
     3. Інвертувати біт режиму роботи через операцію «XOR».
   - Напишіть функцію перевірки статусного біта `int check_status_flag(const uint32_t *reg, uint8_t bit_pos)` , яка приймає захищений покажчик і повертає стан прапорця (0 або 1).
3. Продемонструйте у функції `main` роботу з регістром, виводячи його стан у двійковому або шістнадцятковому вигляді до та після модифікації.
4. Зафіксуйте зміни в Git:

```
gcc -Wall -Wextra task2.c -o task2
git add task2.c
git commit -m "feat(lab5): implement register modification using pointers and bitmasks"
```

Етап 3. Обробка масиву через адресну арифметику (`task3.c`)

1. У каталозі `lab05` створіть файл `task3.c`.
2. Відповідно до свого номера варіанта з **Таблиці 3** реалізуйте алгоритм обробки буфера пам'яті:
   - **Суворе інженерне правило:** У тілі алгоритму **категорично заборонено використовувати квадратні дужки `[]` для індексації**!
   - Уся навігація по масиву має бути реалізована **виключно через арифметику покажчиків**:
     - Використання рухомого покажчика-ітератора на базі інкременту (`ptr++`).
     - Зміщення адреси зі збереженням бази: `*(ptr + i)`.
     - Визначення відстані між покажчиками за допомогою їхнього віднімання (`ptr - start_ptr`), що повертає системний тип `ptrdiff_t`.
   - Забезпечте строгий контроль меж буфера, щоб покажчик не вийшов за кінцеву адресу виділеної пам'яті.
3. Скомпілюйте та зафіксуйте код у Git:

```
gcc -Wall -Wextra task3.c -o task3
git add task3.c
git commit -m "feat(lab5): implement array navigation via pointer
arithmetic without brackets"
```

Етап 4. Долучення звіту та фінальний Push

1. Підготуйте інженерну пояснювальну записку та збережіть її **виключно у форматі PDF** із точною назвою `Report_Lab05.pdf` у каталозі `lab05`.
2. Додайте файл звіту до Git, закомітьте та надішліть усі оновлення на GitHub:

```
git add Report_Lab05.pdf
git commit -m "docs(lab5): add mandatory engineering PDF report"
git push origin main
```

---

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 (Вихідні параметри через покажчики)

Вар.	Вхідні дані та набір результатів, що повертаються через Out-покажчики
1	Вхід: масив чисел. Вихідні параметри: покажчик на мінімальне значення та покажчик на максимальне значення вибірки.
2	Вхід: два цілих числа A і B. Вихідні параметри: покажчик на частку цілочисельного ділення та покажчик на залишок від ділення.
3	Вхід: координати точки (X, Y). Вихідні параметри: покажчик на полярний радіус R та покажчик на полярний кут φ (в радіанах).
4	Вхід: масив вимірювань напруги. Вихідні параметри: покажчик на середнє арифметичне та покажчик на дисперсію значень.
5	Вхід: коефіцієнти квадратного рівняння A, B, C. Вихідні параметри: покажчики на перший (X_1) та другий (X_2) корені; функція повертає кількість дійсних коренів.
6	Вхід: масив цілих чисел. Вихідні параметри: покажчик на кількість парних чисел та покажчик на кількість непарних чисел.
7	Вхід: радіус кола R. Вихідні параметри: покажчик на довжину кола L та покажчик на площу круга S.
8	Вхід: три числа A, B, C. Підпрограма впорядковує ці три змінні за зростанням прямо в пам'яті викликаючої сторони через їхні адреси.
9	Вхід: час у секундах від початку доби. Вихідні параметри: покажчик на години (0..23), покажчик на хвилини (0..59) та покажчик на залишок секунд.
10	Вхід: масив температур. Вихідні параметри: покажчик на індекс найпершого від'ємного значення та покажчик на суму елементів до нього.
11	Вхід: катети прямокутного трикутника A і B. Вихідні параметри: покажчик на гіпотенузу C та покажчик на периметр P.
12	Вхід: ціле додатне число. Вихідні параметри: покажчик на кількість цифр у числі та покажчик на суму його цифр.
13	Вхід: масив опорів резисторів. Вихідні параметри: покажчик на еквівалентний послідовний опір та покажчик на еквівалентний паралельний опір.
14	Вхід: два дробові числа. Підпрограма знаходить більше з них і подвоює його в пам'яті, а менше — ділить навпіл через передані адреси.
15	Вхід: масив символів. Вихідні параметри: покажчик на кількість голосних літер та покажчик на кількість приголосних літер.

Таблиця 2. Варіанти до Завдання 2 (Модифікація 32-бітного регістра через покажчик)

Вар.	Номери бітів регістра `uint32_t *reg`, над якими виконуються операції
1	Встановити в 1 біт 4; скинути в 0 біт 11; інвертувати біт 23; перевірити стан біта 31.
2	Встановити в 1 біт 0; скинути в 0 біт 8; інвертувати біт 16; перевірити стан біта 12.
3	Встановити в 1 біт 7; скинути в 0 біт 3; інвертувати біт 19; перевірити стан біта 2.
4	Встановити в 1 біт 2; скинути в 0 біт 15; інвертувати біт 8; перевірити стан біта 20.
5	Встановити в 1 біт 5; скинути в 0 біт 1; інвертувати біт 14; перевірити стан біта 28.
6	Встановити в 1 біт 10; скинути в 0 біт 6; інвертувати біт 22; перевірити стан біта 3.
7	Встановити в 1 біт 1; скинути в 0 біт 9; інвертувати біт 25; перевірити стан біта 17.
8	Встановити в 1 біт 8; скинути в 0 біт 2; інвертувати біт 13; перевірити стан біта 30.
9	Встановити в 1 біт 12; скинути в 0 біт 5; інвертувати біт 18; перевірити стан біта 7.
10	Встановити в 1 біт 3; скинути в 0 біт 14; інвертувати біт 9; перевірити стан біта 21.
11	Встановити в 1 біт 6; скинути в 0 біт 0; інвертувати біт 15; перевірити стан біта 24.
12	Встановити в 1 біт 15; скинути в 0 біт 7; інвертувати біт 4; перевірити стан біта 18.
13	Встановити в 1 біт 9; скинути в 0 біт 12; інвертувати біт 2; перевірити стан біта 27.
14	Встановити в 1 біт 13; скинути в 0 біт 4; інвертувати біт 11; перевірити стан біта 29.
15	Встановити в 1 біт 14; скинути в 0 біт 10; інвертувати біт 1; перевірити стан біта 26.

Таблиця 3. Варіанти до Завдання 3 (Адресна арифметика над буфером без `[]`)

Вар.	Алгоритмічне завдання над масивом через переміщення покажчика-ітератора
1	Знайти мінімальний елемент масиву, пересуваючи покажчик `ptr++`, та визначити його порядковий номер через різницю покажчиків `(ptr - start)`.
2	Здійснити реверс масиву «на місці», використовуючи два зустрічні покажчики: `left_ptr` рухається вперед, `right_ptr` рухається назад.
3	Підрахувати суму елементів масиву, значення яких перевищують середнє арифметичне, використовуючи адресне зміщення `*(ptr + i)`.
4	Скопіювати елементи з одного буфера в інший, переміщуючи два покажчики паралельно, доки покажчик-джерело не досягне кінцевої адреси.
5	Знайти перший від'ємний елемент у буфері та замінити всі наступні за ним елементи нулями за допомогою інкременту адреси.
6	Порахувати кількість переходів через нуль (зміна знака між сусідніми елементами), аналізуючи адреси `*ptr` та `*(ptr + 1)`.
7	Занулити всі парні за значенням елементи масиву, використовуючи перевірку `(*ptr % 2 == 0)` та перехід `ptr++`.
8	Знайти відстань (в елементах через `ptrdiff_t`) між першим максимальним і першим мінімальним елементами масиву.
9	Виконати циклічний зсув масиву вправо на одну позицію, використовуючи тимчасовий покажчик для зсуву елементів від кінця до початку.
10	Знайти суму чисел, розташованих на парних позиціях пам'яті, використовуючи адресний крок `ptr += 2`.
11	Перевірити, чи є масив симетричним (паліндромом), рухаючи два покажчики від протилежних кінців буфера до центральної адреси.
12	Знайти суму елементів між першим і останнім додатними елементами, зафіксувавши їхні фізичні адреси у два окремі покажчики.
13	Стиснути масив, видаливши нулі: покажчик запису рухається тільки при ненульових значеннях, покажчик читання сканує весь буфер.
14	Помножити кожен від'ємний елемент масиву на -1, переміщуючи покажчик ітератором `ptr++` до граничної адреси `end_ptr`.
15	Знайти два сусідні елементи з мінімальною абсолютною різницею, крокуючи парою покажчиків вздовж усього масиву.

5. ВИМОГИ ДО ЗВІТНОСТІ ТА ЗДАЧІ РОБОТИ

Робота захищається очно в комп'ютерному класі.
Обов'язковою умовою допуску до очного захисту є наявність у репозиторії на GitHub **пояснювальної записки у форматі PDF** (`Report_Lab05.pdf`), яка містить титульний аркуш, індивідуальні формулювання завдань, опис розв'язання конфлікту Git, опис масок і розрахунків, повні 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
├── conflict_test.txt
├── lab01/
├── lab02/
├── lab03/
├── lab04/
└── lab05/
    ├── task1.c
    ├── task2.c
    ├── task3.c
    └── Report_Lab05.pdf
```

Примітка щодо Git: Викладач перевіряє історію комітів за допомогою команди `git log --graph --oneline` на предмет успішного ручного розв'язання конфлікту злиття між гілками `main` та `lab05` .

---

6. КОНТРОЛЬНІ ЗАПИТАННЯ ДЛЯ ОЧНОГО ЗАХИСТУ

1. На скільки фізичних байтів зміститься покажчик на тип `uint32_t` при виконанні операції `ptr + 3` ? Чому ця дія не означає зміщення на три байти?
2. Що фізично повертає операція віднімання двох покажчиків одного типу `ptr2 - ptr1` ? Який системний знаковий тип мови C використовується для збереження цього результату?
3. Чому вираз `array[i]` є абсолютно еквівалентним виразу `*(array + i)` ? Які апаратні команди виконує процесор в обох випадках?
4. Що означає кваліфікатор `const` перед типом покажчика в сигнатурі функції: `void read_data(const int *data)` ? Яку дію забороняє компілятор при спробі звернення до `*data` ?

5. Чому розіменування неініціалізованого («дикого») покажчика є набагато небезпечнішим за розіменування нульового покажчика `NULL`?
6. Що таке конфлікт злиття (Merge Conflict) у системі Git? За яких умов він виникає і чому Git зупиняє автоматичне об'єднання гілок?
7. Що означають службові маркери `<<<<<<< HEAD`, `=======` та `>>>>>>>` у тексті конфліктного файлу?
8. Чому передача масиву у функцію завжди виконується за принципом Zero-Copy і чому всередині функції оператор `sizeof(arr)` видає 8 байтів замість повного розміру масиву?