Лабораторні роботи (ЛР) треба робити не всі, а лише номери, перелічені нижче.
(тут кожен пункт є посиланням на детальніші вимоги до конкретної ЛР)

Частина 1 (SDA-1_Metodichka.pdf):

Частина 2 (SDA-2_Metodichka.pdf):

1.1 РОЗГАЛУЖЕНІ АЛГОРИТМИ

Обов'язково в протоколі має бути (відсутність чогось із переліченого вважається помилкою):

1) Титульна сторінка
2) Завдання на лабораторну роботу + умова завдання за варіантом
3) 2 діаграми (блок-схеми) алгоритмів — по одній на кожен спосіб розв'язку задачі
4) Текст програм (програмний код)
5) Скріншоти тестування програм (щонайменше 3-5)

Не обов'язково письмово відповідати на контрольні питання з методички. За бажанням, Ви можете це зробити, але балів це не додасть. І усно я Вас все одно запитаю.

Діаграми

Блоки, у яких виконуються обчислення, позначаються прямокутником. Паралелограмом позначаються блоки вводу/виводу даних.

У блоках обчислень треба вказувати, за якою формулою виконуються обчислення.

Операції обчислень та операції виводу даних це різні за своєю природою операції, тож не варто об'єднувати їх в один блок на діаграмі.

Виходи з блоків розгалуження мають бути позначені або «0» та «1», або «false» та «true», або «так» та «ні», відповідно до того, який з виходів виконується при справдженні умови, а який — при несправдженні.

Розгалуження алгоритму треба будувати таким чином, щоб одна й та ж умова не перевірялась двічі.

Стрілки

- Переходи між блоками (стрілки) не мають перетинати будь-які блоки.

- Стрілки мають бути спрямовані згори вниз чи вбік, але не знизу вгору (виняток — зворотний зв’язок).
- Стрілки можуть бути вертикальні, горизонтальні та зігнуті під прямим кутом. Діагональні стрілки — неправильно.
- У випадку з прямокутними блоками, стрілки мають виходити (так само як і входити) з середини (в середину) сторони прямокутника, а не з його кута.
- Стрілки, що входять в «Кінець» («End»), краще об’єднувати в одну загальну стрілку, а вже ця стрілка входитиме в «Кінець».

Програми

Програма має відповідати побудованій діаграмі алгоритму. Відповідно, якщо на діаграмі зображено один блок, а в програмі йому відповідають кілька різних операторів на різних гілках алгоритму, це неправильно.

Для обчислення невеликих цілочисельних степенів числа (квадрат, куб тощо) **не доцільно** застосовувати функцію **pow** з бібліотеки Math. Ця функція обчислюється через логарифм, тож її обчислювальна складність суттєво вища, ніж у операції множення.

Мета лабораторної роботи — навчитися писати алгоритми з розгалуженнями.
Тому **не слід** собі спрощувати задачу за допомогою написання власних **функцій** для перевірки, якому проміжку належить ікс:

```
/* Приклад, як не треба робити в цій ЛР */
int condition(int x) {
   if (x > 0)
      return 1;
   if (x > -10)
      return 1;
   else
      return 0;
}

int main() {
   condition(x);
}
```

Вихід з програми (return). В ідеалі, має бути один.
“**Мінімізуйте кількість операцій return у кожній процедурі**. Стає тяжче зрозуміти, як працює процедура, якщо, коли ви читаєте її кінець, ви не здогадуєтеся про можливість того, що десь там вгорі є точка виходу з неї.
Використовуйте *return*, коли це покращує читабельність. У деяких процедурах, щойно ви знаєте відповідь, ви хочете повернути її в головну програму негайно. Якщо процедура визначена у такий спосіб, що вона не потребує жодного підчищення (code cleanup), то якщо значення не повертається негайно, це означає,

що ви мусите писати більше коду.”
— Стів Макконелл. “Досконалий код”

Приклад ситуації, коли є сенс в двох return-ах

```
double calculate_smth(double[] data_array, int arr_length) {
        if (arr_length == 0)
                return -1; // якщо масив має довжину 0, тобто порожній, одразу
                                //виходимо з процедури

        for (int i = 0; i < arr_length; i++)
                // робимо якісь обчислення
        return result;
}
```

Але в першій лабораторній треба всі логічні умови розглянути в тілі основної програми, тому там багато return-ів робити не слід.

Оформлення коду. Хорошим зразком рекомендацій щодо оформлення коду є NASA C Style Guide.
Знайти цей посібник можна за наступним посиланням:
http://mechatronics.me.wisc.edu/labresources/DataSheets/NASA-GSFC_C_Programming_Styles-94-003.pdf

1.2 АЛГОРИТМИ З ВКЛАДЕНИМИ ЦИКЛАМИ ТА МЕТОД ДИНАМІЧНОГО ПРОГРАМУВАННЯ

Обов'язково в протоколі має бути (за відсутність переліченого нижче знімаються бали):

1) Титульна сторінка
2) Завдання на лабораторну роботу + умова завдання за варіантом
3) Текст програм (програмний код)
4) Скріншоти тестування програм (щонайменше 2)
5) *Таблиця з результатами запуску* програм для різних значень *n* = 1, 2, 3, 10, 20, 30, 50, 100.
6) *Результати перевірки* правильності обчислень на калькуляторі (щонайменше 2)

Не обов'язково (за це бали не знімаються, це не оцінюється, тому нема потреби це робити):

1) *діаграми алгоритмів*
2) письмово відповідати на контрольні питання з методички

Обчислення

Для обчислення квадрата чи куба якогось числа **немає потреби застосовувати функцію pow**. Ця функція обчислюється через логарифм, а це, в свою чергу, потребує від комп'ютера більше обчислень, ніж звичайне множення. Варто дбати про оптимізацію обчислень, і застосовувати більш прості операції.

Обчислення мають відбуватися **з високою точністю**.
Тип даних **double** не лише дає змогу зберігати числа з рухомою комою із більшою точністю, ніж float. При обчисленнях (особливо діленні, множенні тощо) у float починає виникати похибка обчислень через неправильне округлення.
double має спеціальний вбудований механізм правильного округлення чисел, і завдяки цьому результат буде значно точнішим.

Перевірка правильності обчислень

Декілька результатів (із тих, які є на скріншотах) слід перевірити на калькуляторі. Треба вхідні дані підставити в формулу й переконатись, що результат збігається. Калькулятор можна обрати будь-який або можна власноруч порахувати самостійно. Як результат у протокол вставити скриншот калькулятора або ж ланцюжок покрокового обчислення результату.

Зважайте, що обчислення мають відбуватися з високою точністю. Якщо похибка виникає вже в другому розряді після коми, а то й лівіше, — отже, десь є якась помилка.

Таблиця

У програмі має бути змінна-лічильник, значення якої збільшується щоразу, як виконуються якісь операції. Після виконання завдання виводиться на екран не лише результат обчислення, а й кількість операцій.

Звіт має містити табличку наступного зразка:

n		1	2	3	10	20	30	50	100	
Кількість операцій	1 спосіб									
	2 спосіб									

1.6 АЛГОРИТМИ ОБХОДУ ДВОВИМІРНИХ МАСИВІВ (МАТРИЦЬ)

Обов’язково в протоколі має бути (за відсутність нижчепереліченого знімаються бали):

1) Титульна сторінка
2) Завдання на лабораторну роботу + умова завдання за варіантом
3) Текст програм (програмний код)
4) Скріншоти тестування програм (щонайменше 5)

Скріншоти

— мають демонструвати коректність послідовного заповнення консолі символами за схемою, заданою за варіантом.

Зверніть увагу

1. Відступи між рядками чи стовпчиками схеми робити не потрібно, тобто, в результаті виконання програми консоль має бути повністю заповнена символами.
2. Відступу згори теж не має бути, перший рядок має бути заповнений.
3. У методичці про роботу з консоллю написано для мови Pascal. У мові С це відбувається дещо інакше (див. конспект лекцій).

2.1 АЛГОРИТМИ ДВІЙКОВОГО ПОШУКУ

Обов'язково в протоколі має бути (за відсутність переліченого знімаються бали):

1) Титульна сторінка
2) Завдання на лабораторну роботу + умова завдання за варіантом
3) Текст програми
4) Вхідні дані
5) Скріншоти тестування програми

Зверніть увагу

1. Пошук має відбуватися на місці, ***додаткові масиви заборонені***.
2. Сортувати масив не потрібно, ***вводити слід вже відсортований масив***. Тож програма має виконуватися з урахуванням того, що на вхід завжди подається відсортований масив.
   (Тобто, уявіть ситуацію: Вам, як програмісту, поставили задачу написати модуль, що шукає число. А сортування буде писати якийсь інший фахівець. Отож, Ви тестуєте свою частину роботи лише з відсортованим масивом).
3. Шуканий елемент у масиві має міститися кілька разів таким чином, щоб видно було, чи алгоритм знаходить випадковий елемент, що співпадає з шуканим (Алгоритм No1), чи знаходить найлівіший з елементів, що співпадають із шуканим (Алгоритм No2).
   Наприклад, якщо масив відсортований за незменшенням, то він може мати вигляд типу 1111222233333444...
4. Алгоритми No1 і No2 — див. конспект лекцій.
5. Незменшення — означає, що кожне наступне число не менше (тобто, більше або рівне даному); незбільшення — кожне наступне число не більше (тобто, менше або рівне даному).
6. Введений масив виведіть у наочному вигляді, себто, у формі матриці (себто, у двовимірному, а не одновимірному вигляді) — щоб було видно, де там який рядок.
7. Множина цілих чисел є лише підмножиною дійсних чисел. Цілі числа це не те ж саме, що дійсні.

   Побічна діагональ починається з нижнього лівого кута.
   https://mathworld.wolfram.com/SkewDiagonal.html

2.2 АЛГОРИТМИ СОРТУВАННЯ

Обов’язково в протоколі має бути (за відсутність переліченого знімаються бали):
Див. розділ “Зміст звіту” в методичці.

Зверніть увагу

1. Сортування має відбуватися на місці, ***додаткові масиви заборонені***.
2. Сортування має бути виконане саме за тим алгоритмом, що вказаний за варіантом.
3. Введений масив, так само, як і відсортований, виведіть у наочному вигляді, себто, у формі матриці (себто, у двовимірному, а не одновимірному вигляді) — щоб було видно, де там який рядок.

   Побічна діагональ починається з нижнього лівого кута.
   https://mathworld.wolfram.com/SkewDiagonal.html