Лабораторний практикум

Viacheslav Davydov (author) Heorgii Molchanov (contributer)

Зміст

Загальні положення та вимоги

Дисципліна “Програмування” викладається два семестри. У кожному семестрі передбачено виконання студентами кількість лабораторних робіт, визначених викладачем.

Усі лабораторні роботи поділени на 6 модулів. 7й модуль - модуль додаткових лабораторних робіт, що може бути використовано для отримання додаткових балів.

За результатами кожної лабораторної роботи студент повинен розробити звіт *державною мовою* обсягом 3–8 сторінок, який складається з таких розділів, що наведені у Вимогах до структурної побудови звіту.

Залежно від викладача, при здачі лабораторної роботи можуть силу наступні штрафи:

- запізнення здачі роботи (більш ніж на тиждень);
- погано оформлений звіт;
- погано оформлений або реалізований код;
- пробіл у теоретичних знаннях щодо теми.

Подяки, люди, що вклали внесок до розвитку курсу дисципліни

- Молчанов Георгій [xone@ukr.net]:
  - ідеологічний ревьювер, sample-project контріб’ютер
- Главчева Дар’я [agafina99@gmail.com]
  - допомога з складанням завдань для комплексної роботи
- Малохвій Едуард [malokhvii.ee@gmail.com]
  - створення проекту автоматизованої перевірки структури директорій (https://git-lab.com/malokhvii-eduard/globlint/)

Варіативність вимог впродовж всього курсу

Слід мати на увазі!! При розробці лабораторних робіт є правило - якщо ви чогось ще не вивчали - не треба це використовувати. Наприклад, не треба використовувати бібліотечні функції до моменту роботи з функціями, або не треба виконувати введення / виведення даних доки ви не дійшли до стосовної лабораторної роботи.

- лабораторні роботи повинні бути виконані у ОС Linux
- звіт з лабораторних робіт та виконаний проект повинні зберігатись у системі контролю версіями *git*
- при створюванні проекту слід використовувати наступні утиліти:
  - для компіляції - *clang*
  - для відлагодження - *lldb*
  - для збірки - *make* з використанням *Makefile*
  - для перевірки якості коду - *clang-format*, *clang-tidy*, *cppcheck*
- упродовж першого семестру розробка коду повинна бути виконана у консолі та за допомогою простих текстових редакторів
- тільки починаючи з лабораторної роботи, що присвячено функціям:
  - можна використовувати бібліотечні функції
  - необхідно використовувати булевий тип з хедер файлу *stdbool.h*
- починаючи з лабораторних робіт, що присвячені схемам алгоритмів та документації, звіт повинен бути оформлений стосовно вимогам, а також з використанням doxygen документації та схемам алгоритмів.
- починаючи з лабораторної роботи, що присвячена вказівникам, необхідно використовувати багатофайлову структуру
- починаючи з лабораторної роботи, що присвячена вказівникам, необхідно використовувати

утиліту *valgind* для відстеження витоків пам’яті

- тільки починаючи з лабораторної роботи, що присвячено вводу/виводу, можна використовувати функції printf/scanf
- починаючи з другого семестру
  - студент має офіційну можливість переходу до IDE
  - необхідно на кожну розроблену функцію писати модульні тести. Перевіряти покриття коду за допомогою функції *llvm-cov*

Модуль 1. Вступ до програмування

Лабораторна робота №1. Вступ до програмування. Освоєння командної строки Linux

Хід роботи.

Дана лабораторна спрямована на установку середовища для подальшої роботі з предмету "Програмування". Тому, ніякий код писати не треба та завдання одне на всіх.

Зверніть увагу! У рамках завдання буде лише загальна інформація, яку треба зробити та, можливо, за допомогою чого. Пошук конкретної команди або послідовність дій треба шукати та знаходити самому.

Загальні рекомендації. Перед тим, як виконувати дії, ознайомтесь, будь ласка, зі всіма пунктами завдання/роботи та, за необхідністю, ознайомтесь з додатковою літературою.

Дії, що необхідно виконати:

1. Знайти, завантажити та інсталювати додаток **VirtualBox**, що дозволить вам встановити віртуальну операційну систему *Linux* та працювати з нею доки ви працюєте з ОС Windows. **Однак,** цей крок можна пропустити, якщо:
   - ваша основна система є *Linux*;
   - ви маєте операційну систему Windows 10 та вмієте встановлювати Windows Subsystem for Linux;
2. Знайти, завантажити та встановити образ *Linux* системи для подальшої роботи. Для кращої роботи також рекомендується встановити *Guest Additions* або використовувати образ з вже встановленими *Guest Additions*. Якщо ваша основна система є *Linux*, то цей крок можна пропустити. Обмежень нема, але існують наступні рекомендації:
   - **не треба проявляти героїзм та інсталювати ОС самому** - існує багато заздалегідь проінстальованих та сконфігурованих образів, що можна завантажити з інтернету, наприклад, сайту *osboxes.org*;
   - методичне забезпечення створювалась з урахуванням того, що у вас буде *Debian*-подібна ОС. Ніхто Вам не забороняє ставити ОС іншої категорії, але виникає ризик того, що проблеми роботи з цією ОС прийде вирішувати Вам самим;
   - якщо у Вас "слабий" комп'ютер (або ноутбук), то рекомендовано ставити *Lubuntu* ОС, або *Linux lite*;
   - якщо у Вас "потужній" комп'ютер (або ноутбук), то рекомендовано ставити *Ubuntu* ОС, або *Kali Linux*;
   - завантажувати має сенс останню доступну ОС;
   - завантажуйте x64-бітну ОС, за винятком, якщо у вас x32 ЕОМ, що не підтримує x64;
   - **Зверніть увагу! Треба** вимикати/перевантажувати віртуальну ОС так само, як ви вимикаєте/перевантажуєте свій домашній ПК. Треба вимикати засобами ОС, а не "клацнувши" на "хрестик"! У Ваших попередниках були випадки, коли після некоректного завершення VirtualBox система більше не завантажувалась і всі незбережені дані знищувались!;
   - якщо ви вирішили використовувати debian-based ОС, то після першого запуску рекомендується оновити компоненти системи до актуальних (відкривши термінал):
     - `sudo apt-get update`;
     - `sudo apt-get upgrade`;
     - `sudo reboot`;
3. **Тут і надалі. Усі дії використовуються за допомогою терміналу**. Інсталювати мінімально-необхідні пакети (програмні додатки): *git*, *gcc*, *clang*, *clang-format*, *clang-tidy*, *tree*, *make*, *check*, *cppcheck*, *valgrind*, *llvm-cov*. Упродовж курсу Ви будете до-інсталювати інші пакети, але ці необхідні, як найменш, у цієї роботі. Для інсталяції пакету необхідно

використовувати *пакетний менеджер apt-get* (для *Ubuntu/Kali Linux) або вбудований менеджер пакетів з графічним інтерфейсом - утиліту ОС, до допомагає інсталювати більшість програмних продуктів ОС без зайвих зусиль.

4. За допомогою системи контролю версіями **git** та команди *git clone* виконати клонування проекту (репозиторія), що знаходиться десь в інтернеті, посилання на який надається кожним викладачем окремо.
5. Ознайомитися з утилітою *tree*. Необхідно зайти в директорію склонованого проекту та завдяки утиліти *tree* необхідно вивести на екран структуру каталогів проекту.
6. Зібрати проект (зкомпілувати). Усі необхідні шаманські дії для компіляції проекту все були зроблені до вас та знаходяться у файлі *Makefile*. Для того, щоб скористатися ним та виконати компіляцію проекту, необхідно:
   - в командній строці зайти до склонованого каталогу;
   - перейти до каталогу, де знаходиться файл *Makefile*;
   - перевірити за допомогою команди *ls* дійсність його наявності;
   - виконав командної строці команду: `make clean prep compile check` вдосконалитись, що помилок немає;
   - за допомогою утиліти *tree* знайти створені файли;
   - перейти до каталогу *dist* та виконати отримані бінарні файли.
7. Виконати (обґрунтовано) будь-які зміни до коду, але при умові, що:
   - наступна компіляція проекту буде виконана без помилок;
   - зміни можна побачити у даних, що виводяться на екран проектом.
8. Перекомпілювати проект (командою `make clean prep compile check`) та продемонструвати зміни, що помітні при виведенні на екран.
9. Виконати зміни в Makefile:
   - додати ціль *all*, яка буде виконувати цілі `clean prep compile check`;
   - перевірити роботу *Makefile*, шляхом виконання команди `make all`;
   - виконати зміни у *Makefile* таким чином, щоб файл *test.bin* не створювався; перевірити роботу виконаних змін;
10. Визначити поточну версію утиліти *clang* та *make*;
11. Дослідити роботу утиліти *man* та описати її призначення;
12. За допомогою команди *git diff* показати виконані зміни у файлах
13. Написати звіт о ході виконання роботи та отриманих результатах. Звіт не повинен мати зображення!

Зверніть увагу!. У рамках першої лабораторної роботи нічого завантажувати на gitlab не потрібно. Необхідно створити файл *lab01.txt* та надати його викладачеві домовленим засобом.

Hint!. Додаток **VirtualBox** має механізм снепшотів (*snapshot*), що дозволяє зберігати поточний стан системи. При наявності помилок в системі можна "відкатити" систему до будь-якого збереженого стану. Рекомендується освоїти цей механізм.

Вимоги до оформлення та сдачі звіту

Звіт повинен складатися з наступних пунктів:

- Автор (ПІБ, група), номер роботи, тема роботи;
- пункти ходу роботи (кількість яких повинна дорівнювати (+-) кількістю пунктів в методичці). Кожен пункт має освітлювати наступні моменти (відподвідати на питання):
  - Що треба зробити?
    - Наприклад, "Для визначення поточної версії clang та make ….."
  - Як треба зробити?
    - Наприклад, "… в командної строці вводимо команди `clang --version` для отримання версії clang, та `make --version` для отримання версії утиліти make"

- Що отримали в результаті?
  - Наприклад, “В результаті отримали наступний результат, що свідчить про те, що версія clang=13.0.0, а версія make=3.81”

```
% clang --version
Apple clang version 13.0.0 (clang-1300.0.29.3)
Target: arm64-apple-darwin20.6.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin

% make --version
GNU Make 3.81
Copyright (C) 2006  Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
This program built for i386-apple-darwin11.3.0
```

Посилання на літературу

- Windows Subsystem for Linux Installation Guide for Windows 10: https://docs.microsoft.com/en-us/windows/wsl/install-win10
- Sample Project: https://github.com/davydov-vyacheslav/sample_project
- Virtual Box: Робота зі снепшотами: https://tavalik.ru/snapshots-in-virtualbox/
- Командна строка Linux для початківців: https://discourse.ubuntu.com/t/the-linux-command-line-for-beginner/14304
- The Linux Command Line by William Shotts: http://linuxcommand.org/tlcl.php
- Сайт з образами Лінукс дистрибутивів: http://osboxes.org/
- Робота з `apt-get`: https://itsfoss.com/apt-get-linux-guide/
- 30 базавих команд `git`: https://habr.com/ru/company/ruvds/blog/599929/

Контрольні питання.

1. Що таке Операційна Система? Яки операційні системи ви знаєте?
2. Які засоби віртуалізації ОС існують?
3. Що таке менеджер пакетів? Які менеджери пакетів існують?
4. Що таке система контролю версіями? Які існують системи контролю версіями?
5. Що таке Makefile? Які його ключові обов’язки? Як його використовувати?
6. Що робить команда *tree*?
7. Як дізнатися перелік файлів та даних, що були змінені за допомогою системи контролю версіями *git*?

Лабораторна робота №2. Вступ до програмування. Основи debug процессу

Хід роботи.

Дана лабораторна спрямована на установку середовища для подальшої роботі з предмету “Програмування”. Тому, ніякий код писати не треба та завдання одне на всіх.

Дії, що необхідно виконати:

1. Зареєструватися (якщо треба) на однієї з загально-доступних систем (рекомендовано), таких як *gitlab*, *github* та створити там приватний репозиторій за допомогою веб інтерфейсу проекту. Репозиторій повинен мати наступну назву: *programing-FNAME*, де:
   - *programing* – Назва курсу (програмування);
   - *FNAME* – транслітероване прізвище студента, відповідно до правил згідно з постановою КМУ 2010р. Наприклад, Прізвище *Давидов* трансформується до *Davydov*. Таким чином, назва репозиторію буде “programing-davydov”. Дії при наявності двох студентів з однаковим прізвищем оговорюються з викладачем окремо.

   За умовчуванням, репозиторій публічний, тобто всі мають доступ до нього. Для того, щоб уникнути умисного витоку вашої інтелектуальної власності (коду, звіту, тощо), **необхідно**:
   - зробити ваш репозиторій приватним;
   - додати вашого викладача у співавтори, щоб він мав змогу перевіряти ваш репозиторій.
2. Склонуватися з створеного репозиторія
3. Скопіювати наступні файли:
   - файл `.clang-format` з проекта `sample_project` до кореня вашого репозиторія
   - файл `.clang-tidy` з проекта `sample_project` до кореня вашого репозиторія
   - файл `.gitlab-ci.yml` з проекта `sample_project` до кореня вашого репозиторія
   - файл `Doxyfile` з каталогу `lab00` проекта `sample_project` до каталогу `lab02` вашого репозиторія
   - файл `Makefile` з каталогу `lab00` проекта `sample_project` до каталогу `lab02` вашого репозиторія
   - файл `README.md` з каталогу `lab00` проекта `sample_project` до каталогу `lab02` вашого репозиторія
   - каталог `src` з каталогу `lab00` проекта `sample_project` до каталогу `lab02` вашого репозиторія
   - каталог `test` з каталогу `lab00` проекта `sample_project` до каталогу `lab02` вашого репозиторія

   Таким чином, у вашому проекті повинні бути наступні елементи

```
├── .clang-format
├── .clang-tidy
├── .gitlab-ci.yml
├── README.md
└── lab02
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── src
    │   ├── lib.c
    │   ├── lib.h
    │   └── main.c
    └── test
        └── test.c
```

4. Зафіксувати зміни (за допомогою команди *git commit*) під назвою “Initial copy of sample_project”. **Зверніть увагу**,
   - при виконанні будь-яких *git* операцій, потрібно знаходитись в каталозі *вашого* проекта;
   - необхідно дослідити роботу команди *git commit*, та внести текст повідомлення у рамках цієї ж команди;
   - перед безпоседенім виконанням команди *git commit* необхідно додати файли до *git* індексу.

Для цього необхідно дослідити команду *git add*

5. Виконати зміни на розсуд викладача, зібрати проект (зкомпілувати) та продемонструвати зміни. За умовчуванням зміни рекомендується виконувати наступні:
- на оцінку “задовільно” - всі ті ж зміни, що буди зроблені у коді у попередній роботі;
- на оцінку “добре” та “відмінно” - додати новій тип Тварини “Людина”, зробити так, щоб він був у списку елементів, що виводиться на екран.
6. Відкрити в відлагоднику (debugger) *lldb* ваш виконуючий файл, зупинитися на будь-якій строці та визначити значення змінних (variables). Слід поекспериментувати з можливостями відлагодника “Step in”, “Step over”, “Step out”. Посилання на офіційну документацію по роботі з додатком. В 90% випадків система команд ідентична до його “батька” gdb, тому рекомендується ознайомитись з його документацією з початку (див. інші посилання у розділі 04 посилань).
7. Зафіксувати зміни за допомогою команди *git commit*.
8. Дослідити режими компіляції з інформацією про відлагодження та без неї; описати різницю.
9. Написати звіт о ході виконання роботи та отриманих результатах. Звіт не повинен мати зображення!
10. Завантажити зміни за допомогою команди *git push*. На поточний час, структура репозиторію, що завантажується на видалений сервер, повинна виглядати наступним чином:

```
├── .git
├── .gitignore
├── .clang-format
├── .clang-tidy
├── .gitlab-ci.yml
├── README.md
├── lab01
│   └── doc
│       └── lab01.txt
└── lab02
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab02.txt
    ├── src
    │   ├── lib.c
    │   ├── lib.h
    │   └── main.c
    └── test
        └── test.c
```

Пояснення до структури:

- *.gitignore* - файл з переліком елементів, що не повинні бути добавлені до системи контролю версіями. До цього переліку належать бінарники та згенеровані файли. Створюється один раз та лежить у корені проекту;
- *.clang-format* - файл коректного форматування коду;
- *.clang-tidy* - файл коректної стилістики коду;
- *README.md* файл загального присвячення всієї дисципліни. Файл повинен мати інформацію про дисципліну та автора роботи;
- *lab02/Doxyfile* - файл конфігурації документування коду;

- *lab02/README.md* файл загального присвячення лабораторної роботи. У рамках курсу повинен мати у собі ПІБ автора, номер лабораторної роботи, назву л/р, та завдання до л/р. Створюється для кожної лабораторної роботи;
- *lab02/dist* - директорія з згенерованими даними. **Не повинна бути в системі контролю версіями** (та повинна бути додана до файлу *.gitignore*). Складається з скомпільованих бінарних файлів та *doxygen* документації (див. інформацію про файл *Doxyfile*);

Зверніть увагу на наявність звіту для лабораторної роботи 1.

Вимоги до оформлення та сдачі звіту

Правила оформлення звіту для цієї лабораторної роботи співпадає з правилами оформлення звіту попередньої л/р

Посилання на літературу

- Транслітерація: http://translit.kh.ua/#lat/passport
- Робота з debugger
  - LLDB Tutorial (official): https://lldb.llvm.org/use/tutorial.html
  - Сайт GNU GDB: https://www.gnu.org/software/gdb/documentation/
  - http://sourceware.org/gdb/current/onlinedocs/gdb.pdf
  - http://www.gdbtutorial.com/tutorial/how-use-gdb-example
  - https://www.cprogramming.com/gdb.html
  - https://www.recurse.com/blog/5-learning-c-with-gdb
  - https://habr.com/ru/post/181738/
- Робота з git
  - Git Book (Official): https://git-scm.com/book/ru/v2
  - .gitignore: http://gitignore.io

Контрольні питання.

1. Що таке процес відлагодження (debug)?
2. Чим відрізняється бінарний файл з та без інформацією про відлагодження?
3. Які базові утиліти необхідні для можливості відлагодження додатків?
4. Для чого необхідна утиліта *lldb*?
5. Як дізнатися поточне значення змінної у відлагоднику?
6. Що таке точка зупинки (breakpoint)? Як її поставити?
7. Що таке стек виклику? Як його подивитися?
8. Що роблять наступні команди: Step In, Step Out, Step Over?

Модуль 2. Основи програмування на мові С

Лабораторна робота №3. Розробка лінійних програм

Структура директорії для лабораторної роботи

```
.
└── lab03
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab03.txt
    └── src
        └── main.c
```

Передмова

Починаючи з цієї лабораторної роботи ви повинні використовувати `Makefile`. Для спрощення вам життя, ви можете використовувати наданий у перший роботі приклад. Але треба виконати його модифікацію:

- так як ви працюєте лише з файлом `main.c` (до лабораторної роботи, що стосується покажчиків), вам потрібно видалити використання файлів `lib.c`, `lib.h`. Використовуючи наданий раніше Makefile це можливо зробити, закоментувавши строку 02 (де декларується змінна LAB_OPTS).

Загальна рекомендація. Не соромтесь фіксувати зміни часто. Якщо Ваша робота над лабораторною роботою займає більше одного дня, заради ваших нервів, наприкінці дня зробіть коміт на завантажте зміни до видаленого репозиторія. Таким чином, якщо у вас щось станеться з віртуальною машиною - ви будете мати мінімум незбережених даних (а заново виконати інсталяцію пакетів лабораторної роботи - займає менш ніж годину без вашої присутності)

Індивідуальні завдання.

Виконати одне завдання з пулу завдань на свій розсуд згідно її складності (що впливає на максимальну оцінку, що може бути отримана за лабораторну роботу). **Зверніть увагу**. Викладач має право надати вам додаткове завдання для виконання.

1. Обчислити об’єм циліндра V з радіусом його основи r і висотою h.
2. Визначити відстань, яка пройдена фізичним тілом за час t, якщо тіло рухається з постійним прискоренням a і має в початковий момент часу швидкість $v0$.
3. За заданим радіусом r визначити довжину окружності l, площу кола S та об’єм кулі V.
4. Відомо, що 1 дюйм дорівнює 2.54 см. Задане значення дюймів перевести в сантиметри та навпаки, для заданого значення сантиметрів визначити його еквівалент у дюймах.
5. Визначити площу та периметр квадрата із стороною a.
6. Кут задано у радіанах. Перевести радіанну міру у градуси.
7. Температура задана у градусах за Цельсієм. Перевести температуру у градуси за Фаренгейтом і навпаки – температуру у градусах за Фаренгейтом – у температуру у градусах за Цельсієм.
8. Знайти периметр і площу рівнобедреного трикутника за умови, що задані координати його вершин. Враховуємо, що основа трикутника паралельна осі OX.
9. Дана довжина ребра куба. Знайти об’єм куба і площу його бічної поверхні.
10. Знайти значення n-го елементу арифметичної прогресії, при заданих значеннях початкового значення та кроку прогресії

11. (*) Дано трьох-значне число. Обчислити значення другої цифри в числі. Наприклад, $x = 456$ -> $y = 5$.
12. (*) Дано десяткове ціле 4-розрядне число. Визначити суму цифр заданого числа.
13. (*) Визначити квадрат відстані у просторі між двома точками *M1* і *M2* із їх заданими координатами $(x1, y1, z1)$ і $(x2, y2, z2)$.
14. (*) Дана сума грошей у гривнях. Перевести гривні в долари, євро, російські рублі. Курси валют (долар, євро, російський рубль) задати в вигляді констант.
15. (**) За заданим опором трьох резисторів *r1*, *r2*, *r3*, які з'єднані паралельно, визначити загальний опір.
16. (**) Визначити число, яке отримане виписуванням у зворотному порядку цифр заданого тризначного числа в десятковій системі числення.
17. (**) Дано дійсне число *a*. Не використовуючи тимчасові змінні та користуючись тільки операціями зсуву, отримати значення: $a * 16$, $a * 1024$ – за одну операцію.
18. (**) Дано 4-розрядне число у системі числення *p* (наприклад, 8). Визначити його еквівалент у десятковій системі числення. На вході система числення повинна бути у діапазоні [2..10].
19. (**) Визначити, у скільки разів перша цифра 3х-значного числа більша, ніж остання. Результат "обрізати" до другого знака після коми. Для поточного завдання "обрізання" включає до себе перевірку того, що отримане число відрізняється від очікуваного не більше, ніж на 0.00001. Наприклад, $x = 123$ -> $y = 1/3 = 0.333333 = 0.330000$.
20. (**) Підрахувати суму чисел у заданому діапазоні. Наприклад, при вхідних даних *50* та *52* повинно бути $50 + 51 + 52 = 153$.
21. Прямокутник заданий координатами своїх вершин (*x1*, *y1*, *x2*, *y2*, *x3*, *y3*, *x4*, *y4*). Знайти периметр *P* і площу *S* прямокутника. Враховуємо, що сторони прямокутника паралельні осям координат.

Додаткові обов'язкові умови виконання робот

- текст програми повинен мати коментарі до коду. Точка входу має також бути документована формулюванням завдання, а також по-пунктного його виконанням;
- структура проекту повинна бути згідно вимогам;
- звіт має містити інформацію про хід виконання завдання, а саме:
  - створення проекту та файлу з вихідним кодом;
  - визначити, де знаходиться точка входу; описати її знаходження, призначення, та чому вона повинна бути одна;
  - запуск програми;
  - зупинка посередині виконання програми за допомогою breakpoints. Навести приклад зміни стану програми "на льоту";
- звіт не повинен мати зображення.

Посилання на літературу

Code convention

- https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard
- https://releases.llvm.org/download.html
- https://github.com/torvalds/linux/blob/master/.clang-format
- https://www.kernel.org/doc/html/v5.14/process/coding-style.html
- https://resources.sei.cmu.edu/downloads/secure-coding/assets/sei-cert-c-coding-standard-2016-v01.pdf

Приклад звіту для лабораторних робот 3-7

Лабораторна робота №3. Розробка лінійних програм
Давидов Вячеслав Вадимович, гр. КІТ-24а
Завдання: визначити об'єм конуса.
Основна частина:

- опис роботи основної функції: об'єм конуса визначається за формулою: $V = 1/3 * pi * r^2 * h$, де pi - константа, h - висота конуса, r - радіус основи конуса.
- перелік вхідних даних:
  - h - висота конуса, позитивне дійсне число (float);
  - r - радіус основи конуса, позитивне дійсне число (float).
- перелік констант:
  - pi - математична константа. Дорівнює значенню 3.14159.
- дослідження результатів роботи програми:
  - V - об'єм конуса. Так як всі оперуємі числа є дійсними, то й результуюча змінна є дійсною. Також, слід визначити, що об'єм конуса не може бути негативним. Тому, її тип - float
  - при значенні h=2.5 та r=12, об'єм конуса повинен складати: $1/3 * 3.14159 * 12 * 12 * 2.5 = 376.9908$
  - для підтвердження коректності роботи програми, зупинено відлагодник на строці з "return 0" та введемо команду "print V". Після вводу команди отримали наступне: (lldb) print V (float) $1 = 376.990814

  Як бачимо, результат майже співпав. Похибка 0.000014 є дозволеною при виконанні операцій з плаваючою крапкою. Подібність результатів говорить про те, що програма працює коректно.

Структура проекту лабораторної роботи:

```
.
└── lab03
    ├── README.txt
    ├── Makefile
    └── src
        └── main.c
```

Висновки: при виконанні лабораторної роботи буди набуті практичні навички створення лінійних програм на мові С, зокрема: визначати типи даних, обчислювати математичні вирази, емулювати операцію зведення до ступеню через операцію множення.

Контрольні питання.

1. З яких розділів складається програма мовою С?
2. Який алгоритм називається лінійним?
3. Запишіть усі можливі оператори, які дозволять значення змінної a збільшити на одиницю.
4. Що таке "проект"?
5. Який порядок створення нового порожнього проекту?
6. Як виконати збірку програми?
7. Який порядок додавання нового файлу до проекту?
8. Як виправити помилку у програмі?
9. Як поставити програму на виконання?
10. Чим консольні програми відрізняються від віконних-програм?
11. Чим змінна відрізняється від константи?
12. Чим відрізняється префіксний запис операції інкременту від постфіксного запису?

Лабораторна робота №4. Розробка програм, що розгалужуються

Структура директорії для лабораторної роботи

```
.
└── lab04
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab04.txt
    └── src
        └── main.c
```

Індивідуальні завдання.

Виконати одне завдання з пулу завдань на свій розсуд згідно її складності (що впливає на максимальну оцінку, що може бути отримана за лабораторну роботу). **Зверніть увагу**. Викладач має право надати вам додаткове завдання для виконання.

1. Знайти мінімальне значення серед заданих трьох змінних.
2. Для заданих речовинних чисел *a*, *b*, *c* визначити корінь квадратного рівняння $ax^2+bx+c=0$, якщо він є і тільки один.
3. При заданому *x* обчислити значення *y* відповідно до формули:

$$y = \begin{cases} x^2 - 1, x > 0 \\ 5x^2 - x + 3, x \leq 0 \end{cases}$$

4. Студенти на іспиті отримують оцінки в системі *ЄКТС* (літери *A*, *B*, *C*, *D*, *E*, *F*). За заданою оцінкою визначити її еквівалент у національній формі (цифровій).
5. Визначити, чи існує трикутник з заданими кутами *a*, *b*, *c*.
6. Знайти максимальне значення серед заданих трьох змінних.
7. Дано два дійсних числа *x* та *g*. При цьому *g* - натуральне число. Визначити *y*:

$$y = \begin{cases} x - g, x > g \\ 12 + (g - x), 0 \leq x \leq g \\ x^2, x < 0 \end{cases}$$

8. Визначити, чи існує ромб з заданими сторонами *a*, *b*, *c*, *d*.
9. Визначити, чи існує прямокутник з заданими кутами *a*, *b*, *c*, *d*.
10. Визначити, чи існує трикутник з заданими сторонами *a*, *b*, *c*.
11. Визначити, чи існує ромб з заданими кутами *a*, *b*, *c*, *d*.
12. При заданих значеннях *x* та *m*, визначити значення *y* відповідно до формули:

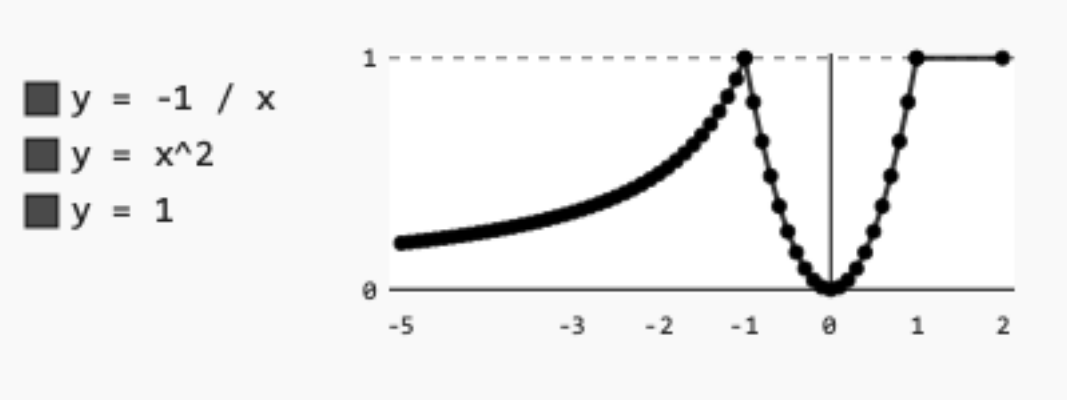
$$y = \begin{cases} x^2 - m^2, x > m \\ 5x^2 - x + 3m, x \leq m \end{cases}$$

13. Визначити, чи існує ромб з заданими сторонами *a*, *b*, *c*, *d*.
14. (*) Визначити, чи є серед цифр заданого трьох-значного числа однакові цифри.
15. (*) Визначити позицію цифри з мінімальним значенням у трьох-значному числі. Нумерація починається з 1. Наприклад, $x = 413$ -> Мінімальне значення дорівнює *1*, що знаходиться у 2й позиції.
16. (*) Визначити століття за заданим роком. Наприклад,
    - $x = 2010$ -> $y = 21$;

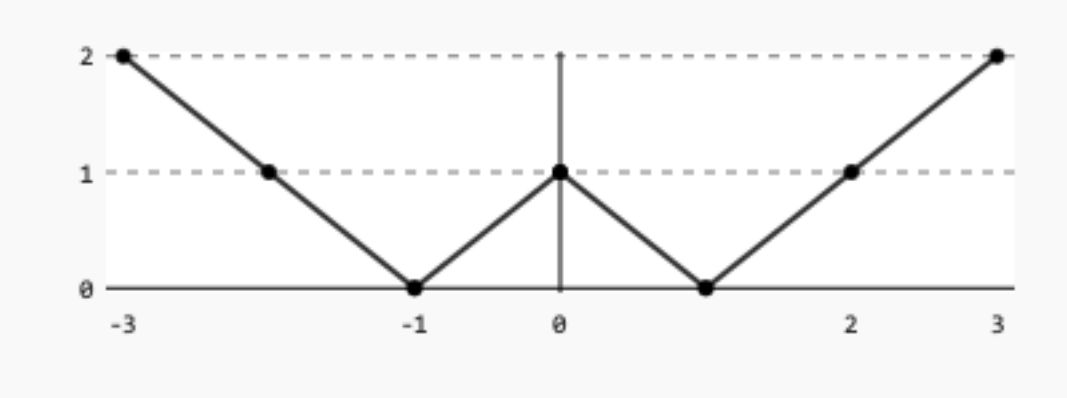
- $x = 2200$ -> $y = 22$.

17. (*) Дано радіус сфери *r* та значення *k*. Використовуючи конструкцію *switch-case* обчислити значення *y* згідно з умовами:
    - $k = 1$. *y* => значення периметри поперечного перерізу;
    - $k = 2$. *y* => значення площі поперечного перерізу;
    - $k = 3$. *y* => значення об'єму сфери;
    - інакше. $y = -1$.
18. (*) Визначити, у скільки разів величина цілої частини числа більше за дробову. Організувати перевірку ділення на 0. Результат "обрізати" до другого знака після коми. Наприклад, $x = 123.656$ -> $y = 123/656 = 0.1875 = 0.180000$.
19. (*) При заданих *a*, *b* та f визначити значення с таким чином:
    - *c* = *a*++ + ++*b*, якщо f дорівнює 1;
    - *c* = *a*-- - --*b*, якщо f дорівнює 0.

    Визначити a та b після обчислень. Обґрунтувати результат.
20. (**) За заданим радіусом *r* та командою (*'l'*, *'s'* або *'v'*) користувача обчислити:
    - довжину окружності, якщо команда – *'l'*;
    - площу кола, якщо команда – *'s'*;
    - об'єм кулі, якщо команда – *'v'*.
21. (**) Визначити, у скільки разів значення дробової частини числа більше за цілу. Організувати перевірку ділення на 0. Результат "обрізати" до другого знака після коми. Наприклад, $x = 123.656$ -> $y = 656/123 = 5.333333 = 5.330000$.
22. (**) Дано три числа *k*, *m*, *n*. Змінити значення змінних таким чином, щоб виконувалась умова $k <= m <= n$. При тестуванні реалізації завдання перевірте комбінації, коли на початку рівняння таке: $k <= m >= n$, $k >= m >= n$, $k >= n >= m$.
23. (**) Дано дійсне число *x*. Для функції *y(x)*, графік якої подано на рисунку нижче, обчислити значення *y*.



24. (**) Дано дійсне число *x*. Для функції *y(x)*, графік якої поданий на рисунку нижче, обчислити значення *y*.



25. (**) Визначити, чи є ціле 6-значне число "щасливим" квитком ("щасливий квиток" – квиток,

в якому сума першої половини чисел номера дорівнює сумі другої половини. Наприклад, білет з номером *102300* є щасливим, бо $1 + 0 + 2 = 3 + 0 + 0$.

Додаткові обов'язкові умови виконання робот

- текст програми повинен мати коментарі до коду. Точка входу має також бути документована формулюванням завдання, а також по-пунктного його виконанням;
- структура проекту повинна бути згідно вимогам;
- звіт має містити інформацію про хід виконання завдання, а саме:
  - створення проекту та файлу з вихідним кодом;
  - визначити, де знаходиться точка входу; описати її знаходження, призначення, та чому вона повинна бути одна;
  - запуск програми;
  - зупинка посередині виконання програми за допомогою breakpoints. Навести приклад зміни стану програми "на льоту";
- звіт не повинен мати зображення.

Контрольні питання.

1. Як працює умовний оператор *if*?
2. Який вираз називається складеним логічним? Наведіть приклади.
3. Який оператор називають оператором множинного вибору? Наведіть приклад.
4. Як працює оператор *switch*?
5. Як працює тернарний оператор? Наведіть приклад.
6. Коли умовний оператор називається вкладеним?
7. Навіщо в операторі *switch* використовується оператор *break*?
8. Чи можуть бути вкладеними оператори *switch*?
9. Чи можна замість оператора *if* використовувати тернарний оператор і навпаки, – замість тернарного – оператор *if*?

Лабораторна робота №5. Циклічні конструкції

Структура директорії для лабораторної роботи

```
.
└── lab05
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab05.txt
    └── src
        └── main.c
```

Індивідуальні завдання.

Реалізувати програму за допомогою трьох типів циклів: *for*, *while-do*, *do-while* (отримати три однакових результати).

Зверніть увагу. У системі контролю версіями (у репозиторію) рекомендується зберігати варіант реалізації тільки одного циклу (обґрунтовано), але викладач має можливість запитати у вас реалізацію завдання з використанням інших циклів.

Виконати одне завдання з пулу завдань на свій розсуд згідно її складності (що впливає на максимальну оцінку, що може бути отримана за лабораторну роботу). **Зверніть увагу**. Викладач має право надати вам додаткове завдання для виконання.

1. Визначити значення $10n!$ (10 * факторіал числа *n*).
2. Визначити кількість щасливих квитків, номери яких складаються з 4 цифр.
3. Визначити суму чисел, які кратні 3 та 5 одночасно, із заданого діапазону. Діапазон задається двома змінними (початок діапазону, кінець діапазону включно).
4. Для заданого числа *x* визначити суму ряду $s = \frac{1}{x} + \frac{2}{x^2} + \frac{3}{x^3} + ... + \frac{10}{x^{10}}$.
5. Підвести число *n* у ступінь *m*.
6. Визначити кількість цифр у заданому цілому числі.
7. У заданому діапазоні цілих чисел визначити останнє число, що ділиться на 7 без залишку.
8. Визначити кількість парних чисел у заданому діапазоні.
9. Знайти добуток усіх чисел із заданого діапазону, що є парними та діляться на 3 без залишку.
10. Визначити суму ряду $s = \frac{1}{x} + \frac{2}{x^2} + \frac{3}{x^3}$..., де *x* – задане число, а останній елемент ряду не повинен бути менший за задане значення числа *epsilon*.
11. Знайти значення *n*-го елементу геометричної прогресії, при заданих значеннях початкового значення та кроку прогресії.
12. Визначити, скільки разів зустрічається задана цифра в заданому числі. Наприклад, в числі 1234231 цифра 3 зустрічається 2 рази, цифра 4 – 1 раз, цифра 5 – 0 разів.
13. Знайти максимальне просте число, що належить діапазону $[a, b]$
14. (*) В заданому цілому числі визначити кількість розрядів та суму його цифр. Наприклад, число *123456* має *6* розрядів, сума його цифр – *21*.
15. (*) Визначити зворотне число для заданого цілого числа. Кількість розрядів від самого початку не визначено. Наприклад, зворотне число для *12334* є число *43321*.
16. (*) Обчислити загальну суму вкладу через *N* років із заданою початковою сумою вкладу та заданою процентною ставкою.
17. (*) Знайти найближче просте число, що більше заданого.
18. (*) Для заданого цілого числа визначити подвійний факторіал. Наприклад:
    - $6!! = 2 * 4 * 6 = 48$,
    - $7!! = 1 * 3 * 5 * 7 = 105$.
19. (*) Знайти найближче просте число, що менше за задане.

20. (⋆) Знайти найбільше 4-значне число, сума цифр якого дорівнює заданому числу.
21. (⋆) Якою мінімальною кількістю купюр можна набрати потрібну суму S за умови, що в наявності є купюри номіналом 1, 2, 5 та 10 грн.?
22. (⋆⋆) Визначити найбільший спільний дільник для двох заданих чисел.
23. (⋆⋆) Визначити, чи є задане ціле число простим.
24. (⋆⋆) Визначити, чи є задане число досконалим (якщо воно дорівнює сумі своїх дільників). Наприклад, 6 - досконале число, бо $6 = 1 + 2 + 3$.

Додаткові обов'язкові умови виконання робот

- текст програми повинен мати коментарі до коду. Точка входу має також бути документована формулюванням завдання, а також по-пунктного його виконанням;
- структура проекту повинна бути згідно вимогам;
- звіт має містити інформацію про хід виконання завдання, а саме:
  - створення проекту та файлу з вихідним кодом;
  - визначити, де знаходиться точка входу; описати її знаходження, призначення, та чому вона повинна бути одна;
  - запуск програми;
  - зупинка посередині виконання програми за допомогою breakpoints. Навести приклад зміни стану програми "на льоту";
- звіт не повинен мати зображення.

Контрольні питання.

1. Як записується і як працює оператор *for*?
2. У чому відмінність оператора *while* від оператора *do ... while*?
3. Як програмуються циклічні алгоритми з явно заданою кількістю повторень циклу?
4. Як програмуються циклічні алгоритми із заздалегідь невідомим числом повторень циклу?
5. Напишіть оператор циклу, який не виконується жодного разу.
6. Напишіть оператор циклу, який виконується необмежену кількість раз.
7. Замініть фрагмент програми з оператором *for* рівнозначним фрагментом програми з оператором *while*.
8. Замініть фрагмент програми з оператором *for* рівнозначним фрагментом програми з оператором *do while*.
9. Як можна перервати виконання оператора циклу?
10. Яке призначення операторів *break* і *continue*?

Лабораторна робота №6. Масиви

Структура директорії для лабораторної роботи

```
.
└── lab06
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab06.txt
    └── src
        └── main.c
```

Індивідуальні завдання.

Виконати одне завдання з пулу завдань на свій розсуд згідно її складності (що впливає на максимальну оцінку, що може бути отримана за лабораторну роботу). **Зверніть увагу**. Викладач має право надати вам додаткове завдання для виконання.

Зверніть увагу. Послідовність (масив) символів не дорівнює строкам (NTCS) !!! Послідовність символів, напр. “ABCD” повинна бути описана як:

```
#define TEXT_LEN 4

char consequence[TEXT_LEN] = { 'A', 'B', 'C', 'D' };
```

1. Генерація нік-нейму (*nick name*): в заданому рядку замінити символи:
   - *a* та *A* на *@*,
   - *o* та *O* на *0*,
   - *i* та *I* на *1*,
   - *s* та *S* на *$*.
2. Відсортувати масив цілих чисел методом “бульбашки” за зростанням або зменшенням залежно від прапору.
3. Замінити усі числа в масиві цілих чисел на значення максимального або мінімального елементу масиву залежно від прапору.
4. Знайти відношення максимального елементу масиву до мінімального. Врахувати, що ділити на 0 не можна.
5. Знайти середнє значення елементів масиву цілих чисел.
6. У масиві цілих чисел визначити суму двозначних чисел.
7. Визначити кількість голосних букв у заданому слові / реченні.
8. Визначити кількість приголосних букв у заданому слові / реченні. При цьому забороняється роботи перелік усіх приголосних букв у блоках *if*, *switch-case*.
9. Дано масив з *k* чисел. З’ясувати, чи відбудеться переповнення розрядної сітки з *n* байтів при визначенні добутку чисел масиву. Наприклад, при $30 * 100 * 30$ -> буде мати місце переповнення розрядної сітки з двох байтів.
10. Для заданої послідовності символів визначити, чи є вона поліандром. Наприклад, A2BCCCCB2A – поліандр.
11. (*) У масиві цілих чисел визначити кількість елементів, що більше попереднього елементу та більше наступного.
12. (*) Заповнити масив цілих чисел заданого розміру числами Фібоначчі.
13. (*) Визначити кількість “щасливих” квитків з 4-розрядними числами (до 9999) та записати їх у масив.
14. (*) Поміняти місцями першу та другу половини заданого рядка. Наприклад,
    - “Ivanov” ==> “novIva”.
15. (*) Знайти перший елемент у масиві, сума цифр якого найбільша в масиві.

16. (*) У заданому тексті знайти слово з найбільшою кількістю букв.
17. (*) Дано двовимірний масив з $N \times M$ цілих чисел. Виконати дзеркальне відображення матриці відносно центрального (якщо кількість стовпців парне - уявного) стовпця.
18. (*) Дано двовимірний масив з $N \times N$ цілих чисел. Попарно поміняти місцями елементи головної та побічної діагоналей.
19. (*) Дано двовимірний масив з $N \times N$ речовинних чисел. Помножити кожен елемент рядка матриці на значення елементу бічної діагоналі відповідного рядка. Повернути кількість рядків матриці.
20. (*) Дано двовимірний масив з $N \times M$ цілих чисел. Повернути з функції середнє значення мінімальних елементів кожного з стовпців матриці.
21. (*) Дано масив з N речових чисел. Необхідно знайти середнє значення даних масиву методом інкрементального середнього.
22. (**) Заповнити масив із заданої кількості елементів простими числами заданого діапазону, що не повторюються. Розмір вихідного масиву задати наперед відомим значенням, що може будуть більшим аніж результуюча кількість отриманих елементів. Якщо масив більше, ніж кількість унікальних значень - невикористовані комірки замінити нулями.
23. (**) Перетворити позитивне число (максимальне значення якого - 9999) в рядок. (усі символи нижнього регістру на виході). Наприклад,
    - 123 – “one hundred twenty three”,
    - 4311 – “four thousands three hundreds eleven”.
24. (**) У заданому тексті знайти кількість слів за умови, що між словами може бути будь-яка кількість пропусків.
25. (**) Дано двовимірний масив з $N \times N$ цілих чисел. Виконати циклічне зрушення елементів рядків масиву в напрямку справа наліво (перший елемент рядка повинен перемістится в її кінець).
26. (**) Дано двовимірний масив з $N \times N$ цілих чисел. Помножити матрицю саму на себе (відповідно до правил множення матриць).
27. (**) Центрувати заданий рядок на площині з із заданим заповнювачем. Наприклад,
    - заповнювач = “_” ,
    - довжина строки = 15,
    - рядок = ”Ivanov \0” (6 символів слово “Ivanov”, 8 - пробілів, останній символ = ‘\0’)
    - результат = “____Ivanov____” (4 символи заповнювача, слово “Ivanov”, 4 символи заповнювача, останній символ = ‘\0’)

Додаткові обов’язкові умови виконання робот

- текст програми повинен мати коментарі до коду. Точка входу має також бути документована формулюванням завдання, а також по-пунктного його виконанням;
- структура проекту повинна бути згідно вимогам;
- звіт має містити інформацію про хід виконання завдання, а саме:
  - створення проекту та файлу з вихідним кодом;
  - визначити, де знаходиться точка входу; описати її знаходження, призначення, та чому вона повинна бути одна;
  - запуск програми;
  - зупинка посередині виконання програми за допомогою breakpoints. Навести приклад зміни стану програми “на льоту”;
- звіт не повинен мати зображення.

Контрольні питання.

1. Як рядки представляються в мові програмування С?
2. Що таке “мірність масиву”? Скільки вимірів може мати масив?
3. Від чого залежить об’єм пам’яті, що необхідний для зберігання елементів масиву?
4. Як можна звернутися до елементу масиву?
5. Як оголошуються масиви?
6. Дайте оцінку наступному оголошенню масиву:

```
int mas[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
```

7. При доступі до елементів двовимірного масиву укажіть, яке призначення першого та другого індексів?
8. Дайте оцінку наступному оголошенню масиву:

```
int mas[][2] = {
    { 1, 2 },
    { 3, 4 },
    { 5, 6 },
    { 7, 8 }
};
```

9. В якому порядку зберігаються у пам’яті комп’ютера елементи двовимірного масиву?
10. Чим характеризуються діагональні елементи багатовимірних масивів? Скільки операторів циклу необхідно для їх обробки?

Лабораторна робота №7. Функції

Структура директорії для лабораторної роботи

```
.
└── lab07
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab07.txt
    └── src
        └── main.c
```

Індивідуальні завдання.

Переробити програми, що були розроблені під час виконання лабораторних робіт з тем “Масиви” та “Цикли” таким чином, щоб використовувалися функції для обчислення результату.

Функції повинні задовольняти основну їх причетність - уникати дублювання коду. Тому, для демонстрації роботи, ваша програма (функція *main()*) повинна мати можливість викликати розроблену функцію з різними вхідними даними.

Слід звернути увагу: параметри одного з викликів функції повинні бути згенеровані за допомогою генератора псевдовипадкових чисел *random()*.

Слід звернути увагу (#2): продемонструвати встановлення вхідних даних через аргументи додатка (параметри командної строки). Обробити випадок, коли дані не передались - у цьому випадку вони матимуть значення за умовчуванням, обраними розробником.

Додаткові завдання складного рівня

1. Реалізувати функцію (та продемонструвати її роботу), що визначає, скільки серед заданої послідовності чисел таких пар, у котрих перше число менше наступного, використовуючи функцію з варіативною кількістю аргументів.
   - Наприклад, при вхідних даних `{ 3, 2, 4, 3, 1 }`, результат повинен бути `3` (тобто наступні пари чисел: `3, 2`, `4, 3`, `3, 1`)
2. Реалізувати функції (та продемонструвати їх роботу), що реалізує находження детермінанту матриці.
   - Приклад прототипу функції: `int get_determinant(int[] matrix, size_t rows, size_t cols)`.

Зверніть увагу!!!: **не можна** передавати двовимірні масиви у якості аргументів функцій! Щоб зробити це - ви повинні перетворити масив до одновимірного (https://en.wikipedia.org/wiki/Row-_and_column-major_order, https://en.wikipedia.org/wiki/Matrix_representation) та передавати його. Після прийому одновимірної інтерпретації двовимірного масиву рекомендується виконувати доступ до `[i][j]` елементу масиву за допомогою формули `[i * N + j]`.

Додаткові обов’язкові умови виконання робот

- текст програми повинен мати коментарі до коду. Точка входу має також бути документована формулюванням завдання, а також по-пунктного його виконанням;
- структура проекту повинна бути згідно вимогам;
- звіт має містити інформацію про хід виконання завдання, а саме:
  - створення проекту та файлу з вихідним кодом;
  - визначити, де знаходиться точка входу; описати її знаходження, призначення, та чому вона повинна бути одна;
  - запуск програми;

  - зупинка посередині виконання програми за допомогою breakpoints. Навести приклад зміни стану програми “на льоту”;
- звіт не повинен мати зображення.

Контрольні питання.

1. Чому при передачі параметра за значенням усі зміни параметра у функції не відбиваються на значенні аргументу?
2. Скільки операторів *return* може бути в тілі функції?
3. Що таке “прототип функції”, яке його призначення?
4. Як описати функцію, яка не має значень, що повертаються?
5. Як описати, що функція не має аргументів?
6. Наведіть приклад функції, яка не має аргументів та нічого не повертає.
7. Що робить функція *random()* / *rand()*? Чому генератор надає “псевдо-випадкові” числа?
8. Як забезпечити генерацію іншої послідовності чисел при кожному наступному виконанні програми?
9. Чим відрізняються бібліотечні функції від функції користувача?
10. Які функції називаються варіативними?
11. Як визначити список параметрів функції змінної довжини? Наведіть приклад.

Модуль 3. Основи документування

Лабораторна робота №8.1. Вступ до документації коду

Загальне завдання

Для лабораторної роботи “Функції” необхідно додати можливість генерації Doxygen документації.

Перелік пакетів/утиліт, що повинні бути встановлено в контексті цієї роботи:

- doxygen;
- mscgen;
- graphviz.

Починаючи з цієї роботи Makefile повинен генерувати Doxygen документацію (наприклад, додав ціль `doxygen`). У звіті секції опису даних повинні бути скопійовані з результатів Doxygen документації.

У ході виконання роботи, ви повинні отримати наступну структуру.

```
.
└── lab08
    ├── README.md
    ├── Doxyfile
    ├── Makefile
    ├── src
    │   └── main.c
    └── dist/
        └── html/
```

Пояснення до структури:

- у ході виконання команди `make doxygen`, в директорії dist повинна з’явитися директорія html. Зміст цієї директорії не повинен понрапити до системи контроля версіями;
- як і у попередніх роботах, зміст директорії dist не повинен бути зафіксовано у системі контролю версіями;
- файл `main.c` повинен бути скопійований з лабораторної роботи “Функції”. В файл повинно бути додано `doxygen` документацію.

Зверніть увагу. На головної сторінці *doxygen* звіту повино бути розташотано контекнт файлу `README.md`.

Посилання на літературу

- Ефективне документування коду за допомогою Doxygen: https://habr.com/ru/post/252101/

Контрольні питання.

1. Що таке система *Doxygen*?
2. Які результуючі формати документації підтримує *Doxygen*?
3. Як у консольному режимі згенерувати документацію до проекту? Що для цього потрібно?
4. Призначення утиліти *dot*.
5. Призначення пакету *mscgen*.
6. Як у коді визначити, що поточний коментар потрапить до документації?
7. Які команди необхідні для документування файлу взагалі?
8. Які механізми існують для документування загальної інформації про проект?

Лабораторна робота №8.2. Вступ до документації проекту

Загальне завдання

Розробити повноцінний звіти для лабораторної роботи “Функції”, що присвячена функціям у двох форматів (+їх репрезентація у PDF форматі):

- *Markdown*.
- *doc* формат, згідно ДСТУ.

Детальну інформацію можна отримати за посиланнями у стосовних підрозділів розділу “Перелік літератури”

За складом, обидва звіту повинні повністю задовольнять вимогам оформлення звітів, що описаний у початку розділу лабораторних робіт першого семестру.

Починаючи з цієї роботи звіт має бути оформленим у Markdown стилі або у doc форматі згідно ДСТУ (за попередньою домовленістю з викладачем), згідно декларованим вимогам.

У ході виконання роботи, в директорії `lab08` повина бути додана папка `doc` наступної структури:

```
.
└── lab08
    ....
    └── doc
        ├── lab08.docx
        ├── lab08.pdf
        └── lab08.md
```

Пояснення до структури:

- lab08.pdf - pdf репрезентація файлу lab08.docx. Щоб зробити з docx файлу pdf - треба в MS Word / Libre Writer у меню “Зберегти Як …” вибрати тип “PDF Document”;
- зображення для lab08.md (наприклад, схеми алгоритмів розроблених функцій) повинні бути посиланням на директорію assets лабораторної роботи “Схеми Алгоритмів”. При подальшому виконанні завдань, директорія assets повинна розташовуватися в каталозі doc того ж самого проекту.

Зверніть увагу!!!. Усі подальші лабораторні роботи повинні бути виконано з використанням Markdown

Вимоги до оформлення текстової частини звіту (markdown формат)

- назва кожного пункту меню повинна бути виділена жирним стилем
- усі рисунки повинні бути пронумеровані та мати опис та посилання
- текст коду (у тому числі його частини), структура проекту та результат роботи (якщо програма щось виводить на екран) повинні бути відображені у моноширинному стилі завдяки спец-символам ‘

Зверніть увагу! У зв’язку з тим, що текст (raw text) у Markdown файлі проходить обробку та є відмінності від результуючого pdf файлу, велике прохання перед тим як видавати викладачу цей файл, спочатку його пере-прочитати на наявність стилістичних розбіжностей !!!

Вимоги до структурної побудови звіту Формат звітів буде трансформуватися з ходом виконання лабораторних. Перші лабораторні роботи мають формат лише звітів. Формат звітів вільний, але викладач має право ввести свої обмеження.

Потім додається схематичний опис розроблених модулів, використання системи автоматичної генерації коду, дотримання ДСТУ та ін.

Звіт складається з розділів. Для кожного розділу вказується номер та його назва. Обов’язковими складовими звіту є:

- **Номер роботи та її тема**. Ці пункти не нумеруються. Вказується з вирівнюванням по центру рядка.

- **ВИМОГИ**:
  - *Розробник*. Інформація про розробника: чи є студентом, прізвище, ім'я, по батькові; назва академічної групи; номер варіанту; дата розробки програми.
  - *Загальне завдання*. Зауваження, обмеження та вимоги.
  - *Індивідуальне завдання*. Прикладна задача відповідно до варіанта.
- **ОПИС ПРОГРАМИ**. Цей підрозділ відповідає за внутрішню роботу програми, її поведінку: калькулювання даних, обробка даних та інші функції програми:
  - *Функціональне призначення*. Призначення програми. Обмеження на застосування.
  - *Опис логічної структури*:
    * Якщо використовується об'єктно-орієнтований підхід, навести UML – схему (рисунок), що відображає внутрішню структуру і взаємозв'язки (відносини) розроблених класів, який також створюється засобами *doxygen*.
    * Описати призначення та описати структуру розроблених методів (в тому числі, їх аргументам) та структур / класів, констант та змінних. Матеріалі для даного підрозділу рекомендується брати з результатів згенерованих doxygen документацій. При цьому слід тільки змінити стилі (шрифт, розмір, відступи, колір та інше) і, при необхідності, перевести на українську мову.
    * Навести структуру програми. У даному підрозділі достатньо зробити скріншот оглядача рішень з списком файлів та методів. При консольному виконанні - достатньо виконати команду *tree* в корені проекту, але перед тим необхідно бути завіреним, що усі генеровані дані (бінарники, документація) видалені. Також, структуру програми можна отримати, зробивши скріншот сторінок *doxygen* документації, що мають перелік файлів та методів.
  - Важливі фрагменти програми. Частини тексту програми, що демонструють рішення задачі та короткий (1-2 речення) опис. Обов'язково повинні бути вказані початкові дані.
- **ВАРІАНТИ ВИКОРИСТАННЯ**. Опис поведінки програми ("хто" і "що" може зробити). Відповідає функціональним вимогам. Ілюструється за допомогою копій екрану з описом. У даному пункті необхідно проаналізувати отримані результати. У разі наявності виведення результатів на екран необхідно навести скріншоти консолі. Якщо робота виконувалася без виведення результатів на екран, подати скріншот вікна відлагоджувача з результатами роботи програми.
- **ВИСНОВКИ**. Даний пункт не нумерується. Необхідно підвести підсумки і зробити висновки про досягнення мети лабораторної роботи (на основі її теми та завдання) та щодо коректності результатів тестування розробленої програми.

Зверніть увагу! Наведений перелік розділів є рекомендований для викладачів. Тому, за викладачем, що веде лабораторні роботи остається останнє слово за остаточним переліком.

Зверніть увагу. В звіті не потрібно наводити весь код програми! Замість цього краще навести посилання на проект в інтернеті (Github, тощо).

Декілька теорії! Функціональні вимоги (Functional Requirements) - це вимоги до програмного забезпечення, які описують внутрішню роботу системи, її поведінку: калькулювання даних, маніпулювання даними, обробка даних та інші специфічні функції, які має виконувати система. На відміну від нефункціональних вимог, які визначають якою система повинна бути, функціональні вимоги визначають, що система повинна робити. Функціональні вимоги до програмного забезпечення визначаються на першій стадії процесу розробки ПЗ — на етапі аналізу вимог.

Посилання на літературу

- Markdown guide: https://www.markdownguide.org

- СТЗВО-ХПІ-3.01-2018 ССОНП. Текстові документи у сфері навчального процесу. Загальні вимоги до виконання. – (http://blogs.kpi.kharkov.ua/v2/metodotdel/standarti-ntu-hpi/)
- Утиліта draw.io: https://www.diagrams.net
- Offline markdown editors:
  - https://wereturtle.github.io/ghostwriter/
  - https://remarkableapp.github.io/linux.html
  - http://pad.haroopress.com/user.html#download

Контрольні питання.

У цієї лабораторній роботі контрольні питання відсутні.

Модуль 4. Сучасні технології мови С

Лабораторна робота №9. Модульне тестування

Структура директорії для лабораторної роботи

```
└── lab09
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── test
    │   └── test.c
    └── src
        ├── lib.c
        ├── lib.h
        └── main.c
```

Зверніть увагу. Для роботи з тестами вам потрібно: - Додати файл з тестами (файл `./test/test.c`) - інсталювати утиліту libcheck (див. https://libcheck.github.io/check/web/install.html) - В Makefile додати цілі для компіляції та запуску тестів (файл `./test/test.c`) з доданням опції `-lcheck`. Наприклад (код взято з sample_project):

```
compile: main.bin test.bin

test.bin: test/test.c
    $(CC) $(C_OPTS) $< $(ARCH) -fprofile-instr-generate -fcoverage-mapping -o ./dist/$@ -lcheck

test: clean prep compile
    LLVM_PROFILE_FILE="dist/test.profraw" ./dist/test.bin
    llvm-profdata merge -sparse dist/test.profraw -o dist/test.profdata
    llvm-cov report dist/test.bin -instr-profile=dist/test.profdata src/*.c
    llvm-cov show dist/test.bin -instr-profile=dist/test.profdata src/*.c --format html > dist/coverage.html
```

Основне завдання

- Переробити проект на багатофайлову структуру, в якої:
  - `main.c` - склаадється з тестового запуску функцій для завідомо відомих даних
  - `lib.c` - складається з реалізації розроблених функій
  - `lib.h` - складається з прототипів розроблених функцій, що документовані
- Для попередньо розробленої функції (функцій) додати методи – модульні тести, що демонструють коректність роботи розробленого функціоналу. Розроблені методи мають перевірити коректність функціонування функцій на наборі заздалегідь визначених вхідних-вихідних даних.

Приклад файл-тесту для веріфікації додатку 2 чисел (функция `int sum(int a, int b);`) (без коментарів):

```
#include "lib.h"
#include <check.h>


START_TEST(test_sum)
{
#define DATA_SIZE_SUM 3
    int input_data_a[] = { 1, 3, -1 };
    int input_data_b[] = { 2, 0, 10 };
    int expected_values[] = { 3, 3, 9 };
```

```
    for (int i = 0; i < DATA_SIZE_SUM; i++) {
        int actual_value = sum(input_data_a[i], input_data_b[i]);
        ck_assert_int_eq(expected_values[i], actual_value);
    }
}
END_TEST

int main(void)
{
    Suite *s = suite_create("Programing");
    TCase *tc_core = tcase_create("lab09");

    tcase_add_test(tc_core, test_sum);
    suite_add_tcase(s, tc_core);

    SRunner *sr = srunner_create(s);
    srunner_run_all(sr, CK_VERBOSE);
    int number_failed = srunner_ntests_failed(sr);
    srunner_free(sr);

    return (number_failed == 0) ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

Зверніть увагу. Дана лабораторна робота не передбачає наявності звіту

Посилання на літературу

- https://clang.llvm.org/docs/SourceBasedCodeCoverage.html

Контрольні питання.

1. Що таке модульний тест?
2. Як протестувати функцію конкатенації двох рядків?
3. Як протестувати функцію об’єднання двох масивів?
4. Як запустити модульний тест?
5. Коли виконуються модульні тести?
6. Які принципи модульного тестування?
7. Навіщо потрібні модульні тести? Які ще бувають тести?

Модуль 5. Основи роботи з показчиками

Лабораторна робота №10. Вступ до показчиків

Структура директорії для лабораторної роботи

```
└── lab10
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab10.md
    ├── test
    │   └── test.c
    └── src
        ├── lib.c
        ├── lib.h
        └── main.c
```

Зверніть увагу!. При роботі з пам'яттю можуть виникати витоки пам'яті. Для дослідження цих витоків слід використовувати утиліту *valgrind*, додав та використовував (`make leak-check`) наступну ціль:

```
V_FLAGS=--tool=memcheck --leak-check=full --show-reachable=yes \
    --undef-value-errors=yes --track-origins=no --child-silent-after-fork=no \
    --trace-children=no --error-exitcode=1


leak-check: clean prep compile
    valgrind $(V_FLAGS) --log-file=dist/valgrind.log --xml-file=dist/valgrind.xml --xml=yes dist/main.bin
    valgrind $(V_FLAGS) --log-file=dist/valgrind.log --xml-file=dist/valgrind.xml --xml=yes dist/test.bin
```

Індивідуальні завдання.

Виконати одне завдання з пулу завдань на свій розсуд згідно її складності (що впливає на максимальну оцінку, що може бути отримана за лабораторну роботу).

Зверніть увагу. Викладач має право надати вам додаткове завдання для виконання.

Зверніть увагу. При виконанні завдання ви повинні використовувати динамічне виділення пам'яті для масивів.

1. Дано масив з N цілих чисел. Знайти мінімальний та максимальний елементи масиву. Визначити суму елементів, що розташовані між цими елементами. Створити другий масив, що містить ці елементи.
2. Дано два масиви: *mas1[N]* і *mas2[M]*. Створити третій масив, у який переписати елементи масиву *mas1*, а потім *mas2*. Отриманий масив упорядкувати за зростанням.
3. Дано два масиви: *mas1[N]* і *mas2[M]*. Створити третій масив, у який переписати по черзі по два елементи з вхідних масивів; почати з масиву *mas2*.
4. Дано масив з N речовинних чисел. Розмістити всі елементи з позитивними значеннями в лівій частині масиву, елементи з негативними значеннями – у правій, а нулі – між ними.
5. Дано масив з N цілих позитивних чисел. Визначити, які числа в масиві є довершеними (такими, що дорівнюють сумі своїх дільників), та перенести їх до нового масиву.
6. Дано масив з N речовинних чисел. Створити другий масив, який буде мати у собі усі елементи початкового масиву, що розташовані між першим та другим негативними елементами.
7. (*) Дано масив з N цілих чисел. Визначити кількість пар сусідніх елементів з однаковими

значеннями. Ці пари переписати у другий масив. Зверніть увагу: 3 послідовних числа - це одна пара та один елемент “без пари”.

8. (*) Дано масив масивів з *N* * *N* цілих чисел. У кожному рядку масиву знайти кількість парних додатних чисел. Отримані результати записати в одновимірний масив.
9. (*) Даний масив масивів з *N* * *N* речовинних чисел. Максимальні елементи кожного стовпця переписати в одновимірний масив.
10. (*) Дано масив масивів з *N* * *N* цілих чисел. Елементи головної діагоналі записати в одновимірний масив, отриманий масив упорядкувати за зростанням.
11. (**) Дано масив з *N* цілих чисел. Визначити, чи є в масиві елементи, що повторюються; якщо такі є, то створити масив, в якому вказати, скільки разів які елементи повторюються. Таким чином, в результуючому масиві кожен непарний елемент - число, що повторюються; кожен парний елемент - кількість повторювань.
12. (**) Дано масив з *N* цілих чисел. Знайти безперервну послідовність позитивних чисел у вхідному масиві, сума елементів якої максимальна, та переписати їх у вихідний масив.
13. (**) Дано масив з *N* речовинних чисел. Підрахувати кількість ділянок, які утворюють безперервні послідовності чисел з не-зменшуваними значеннями. Максимальну ділянку переписати у інший масив.
14. (**) Дано масив масивів з *N* * *N* цілих чисел. Елементи головної діагоналі записати в одновимірний масив, отриманий масив упорядкувати за зростанням.

Додаткові обов’язкові умови виконання робот

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- звіт повинен бути оформлений згідно “Вимогам до структурної побудови звіту”;
- продемонструвати відсутність витоків пам’яті за допомогою утиліти *valgrind*;
- доступ до елементів масиву здійснювати через розіменування покажчиків, а не через оператор індексування (*[ ]*);
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами.
- (як і раніше) забороняється використовувати функції введення/виведення. Вхідні дані повинні бути у вигляді констант, вихідні дані повинні бути відображені за допомогою відлагодника.

Контрольні питання.

1. Як створити покажчик на масив?
2. Які операції можуть бути застосовані до покажчиків?
3. Як здійснюється звільнення пам’яті?
4. Як здійснюється виділення пам’яті?
5. Як створюється контроль за витоком пам’яті?
6. Чим відрізняється статичний масив від динамічного?
7. Як визначити поточний обсяг пам’яті для динамічного масиву?
8. Що виконується в наступному фрагменті програмного коду?

```
pointMas = &mas[0];

for(i = 0; i < arraySize; *pointMas++ = i++);
```

9. Чому треба звільняти пам’ять, яка динамічно виділялась?

Лабораторна робота №11. Взаємодія з користувачем шляхом механізму введення/виведення

Структура директорії для лабораторної роботи

```
└── lab11
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── assets
    │   └── input.txt
    ├── doc
    │   └── lab11.md
    ├── test
    │   └── test.c
    └── src
        ├── lib.c
        ├── lib.h
        └── main.c
```

Зверніть увагу. Початковий Текст зберігається у файлі `assets/input.txt` однією строкою. Програма повинна зчитати дані з стандартного потоку `stdin`. Приклад передачі контенту файлу до стандартного потоку вводу `main.bin`:

```
cd lab11
cat ./assets/input.txt | ./dist/main.bin
```

Індивідуальні завдання.

Виконати завдання з пулу завдань на свій розсуд згідно її складності (що впливає на максимальну оцінку, що може бути отримана за лабораторну роботу). **Зверніть увагу**. Викладач має право надати вам додаткове завдання для виконання.

При виконанні завдань слід зазначити, щоб:

- при старті програми виводилась інформація об авторі, номері та темі лабораторної роботи;
- при запиті даних, користувач отримав повідомлення, що від нього очікують;
- початкові дані вводилися з клавіатури за допомогою функції *scanf()*.;
- видача результуючих даних проводилася у консоль за допомогою функції *printf()*;
- після запиту даних треба виконати валідацію введених даних з точки зору бізнес логіки;
- на базі введених даних створити стосовні дані *речових* чисел в діапазоні `(-1000; 1000)` з мінімальним кроком 0.001
- результат роботи виводу матриці повинен бути відформатованим стосовно наступного прикладу:

```
[   -5.25  33.25    23.00 ]
[ -454.50   0.00   123.23 ]
[  323.99 123.55  -123.00 ]
```

Завдання:

1. Визначити суму двох матриць.
2. Визначити добуток двох матриць.
3. Визначити транспоновану матрицю.
4. (*) Сформувати трикутник Паскаля ітеративним та рекурсивним методом.
5. (*) Визначити зворотню матрицю.

Додаткові завдання підвищеної складності:

1. (**) Виконати індивіджуальне завдання з взаємодією з користувачем шляхом використання функцій *write()*, *read()*

Додаткові обов'язкові умови виконання робот

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- звіт повинен бути оформлений згідно "Вимогам до структурної побудови звіту";
- продемонструвати відсутність витоків пам'яті за допомогою утиліти *valgrind*;
- доступ до елементів масиву здійснювати через розіменування покажчиків, а не через оператор індексування (*[ ]*);
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами.

Контрольні питання.

1. Як виводити текст на консоль?
2. Як читати дані з клавіатури?
3. В чому різниця форматованого виведення даних від неформатованого?
4. У якому файлі знаходяться прототипи функцій введення/виведення?
5. Який символ використовується для формування адреси змінної?
6. Як можна форматувати дані при їх виведенні на екран?
7. Які дії виконує функція читання при введенні даних з клавіатури?
8. Які дії виконує функція виведення даних на екран?
9. Як ввести текст, який складається з декількох слів?
10. Як вивести текст, який складається з декількох слів?

Лабораторна робота №12. Строки (Null-terminated C Strings)

Структура директорії для лабораторної роботи

```
└── lab12
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── assets
    │   └── input.txt
    ├── doc
    │   └── lab12.md
    ├── test
    │   └── test.c
    └── src
        ├── lib.c
        ├── lib.h
        └── main.c
```

Зверніть увагу. Початковий Текст зберігається у файлі `assets/input.txt` однією строкою. Програма повинна зчитати дані з стандартного потоку `stdin` (наприклад, за допомогою функції `fgets`). Приклад передачі контенту файлу до стандартного потоку вводу `main.bin`:

```
cd lab12
cat ./assets/input.txt | ./dist/main.bin
```

Розмір буферу повинен бути завідомо відомий, наприклад, 1000 символів.

Індивідуальні завдання.

Виконати одне завдання з пулу завдань на свій розсуд згідно її складності (що впливає на максимальну оцінку, що може бути отримана за лабораторну роботу). **Зверніть увагу**. Викладач має право надати вам додаткове завдання для виконання.

1. Визначити, скільки у тексті оповідальних, питальних, окличних речень (без використання ітерації по кожному символу у циклу).
2. Визначити кількість слів у кожному рядку тексту.
3. Визначити, скільки разів у тексті зустрічається задане слово, яке необхідно ввести з клавіатури.
4. Визначити, скільки у тексті голосних і скільки приголосних букв. Вивести на екран процентне співвідношення.
5. Текст – це програма на мові *C*. Визначити, скільки в ньому операторів циклу.
6. Текст – це програма на мові *C*. Визначити, чи є у наведеному тексті всі пари дужок: *()*, *{}*, *[]*. При реалізації цього завдання необхідно створити генералізовану функцію для обробки.
7. Визначити кількість таких слів у тексті, у яких перший і останній символи збігаються між собою.
8. У кожному рядку тексту змінити порядок символів на протилежний.
9. Вилучити з тексту всі пробіли на початку і зайві пробіли між словами, залишивши по одному.
10. Вивести саме довге та саме коротке слово з тексту.
11. (⋆) Усі скорочення (т.д., т.п., ін.) замінити на повні словосполучення.
12. (⋆) Знайти найдовше та найкоротше слово в заданому тексті.
13. (⋆) У тексту записана (без помилок) така послідовність символів: $a\#b$, де a і b – цілі числа, $\#$ – одна з арифметичних операцій. Наприклад, $17 + 2$.
14. (⋆) “Зашифрувати” вхідний текст шифром “Цезаря”.

15. (*) “Зашифрувати” вхідний текст, для чого в кожному рядку тексту поміняти місцями кожний символ на символ, що знаходиться на відстані від поточного на N позицій. Виконати дешифрування.
16. (*) Для запам’ятовування числа π іноді використовують наступну російську фразу: “это я знаю и помню прекрасно Пи многие знаки мне лишни напрасны”. Число букв у кожному слові – це наступна цифра числа: “это” – 3, “я” – 1, “знаю” – 4 і т. д. Розробити програму, яка за зазначеним алгоритмом буде видавати число, використовуючи будь-який текст.
17. (**) Визначити, скільки у тексті слів (без використання ітерації по кожному символу у циклу). Видати всі слова за абеткою.
18. (**) Текст – це перелік прізвищ студентів через кому. Видалити з тексту усі дублікати.
19. (**) Вирахувати для тексту частотну таблицю: для кожного символу визначити його частоту появи у тексті (число таких символів у тексті ділене на загальне число символів у тексті).
20. (**) Знайти всі числа, які зустрічаються в тексті.
21. (**) Сформувати частотну таблицю символів у тексту та вивести її на екран (з вказанням кількості та процентного відношення). Обмеження - виводити тільки ті символи, що зустрічаються у тексті.
22. (**) Без використання зовнішніх функцій необхідно реалізувати функції перетворення строки, що представляє собою число до фактичного числа (supported types: int, float). У рамках завдання можемо вважати, що строка не має помилок та має лише число

Додаткові обов’язкові умови виконання робот

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- звіт повинен бути оформлений згідно “Вимогам до структурної побудови звіту”;
- продемонструвати відсутність витоків пам’яті за допомогою утиліти *valgrind*;
- доступ до елементів масиву здійснювати через розіменування покажчиків, а не через оператор індексування (*[ ]*);
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами.

Контрольні питання.

1. Як “склеїти” два рядки?
2. Як визначити, чи є в заданому рядку заданий підрядок?
3. Чому рядки в мові С мають назву “Null-terminated C Strings” ?
4. Як порівняти два рядки?
5. Як у заданому рядку видалити заданий підрядок?

Лабораторна робота №13. Взаємодія з файлами

Структура директорії для лабораторної роботи

```
└── lab13
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── assets
    │   └── input.txt
    ├── test
    │   └── test.c
    ├── doc
    │   └── lab13.md
    └── src
        ├── lib.c
        ├── lib.h
        └── main.c
```

Зверніть увагу. Вхідні дані розташовані у файлі `assets/input.txt`. Програма повинна прийняти у якості аргументу шлях до цього файлу. `output.txt` - файл для запису результатів. Приклад передачі файлу у якості параметра додатка `main.bin`:

```
cd lab13
./dist/main.bin "./assets/input.txt" "./dist/output.txt"
```

Розмір буферу повинен бути визначений через визначення розміру файлу з даними.

Загальне завдання

- початкові дані вводилися з файлу;
- видача результуючих даних провадилася не тільки у консоль, але й у файл (але його не повинно бути у системі контролю версіями).
- ім'я вхідного файлу та результуючого фалу повинно отримано від користувача за допомогою аргументів додатку;
- робота з файлом повинна бути використована за допомогою функцій *fprintf()* та *fscanf()*;
- очікується, що при запуску програми з параметрами командної строки, нічого від користувача не буде очікуватись.

Рекомендації щодо (приклад) прототипів функцій роботи з файлами:

```
void read_from_file(FILE *f, char* result, int buf_size);
void write_to_file(FILE *f, char* data);
```

Індивідуальні завдання.

Виконати одне завдання з пулу завдань на свій розсуд згідно її складності (що впливає на максимальну оцінку, що може бути отримана за лабораторну роботу). **Зверніть увагу**. Викладач має право надати вам додаткове завдання для виконання.

1. Розробити програму, що в заданому файлі створює суцільний рівнобедрений трикутник із заданих символів (наприклад, зірочок). Висота трикутника, символ заповнювача та ширина основи задається у вхідному файлу. Наприклад, при висоті 4, ширині 7 та заповнювачі `*` результат у файлі потрібен бути наступним:

```
   *
  * * *
 * * * * *
* * * * * * *
```

2. Розробити програму, що в заданому файлі створює порожнистий рівнобедрений трикутник із заданих символів (наприклад, зірочок). Висота трикутника, символ заповнювача та ширина основи задається у вхідному файлу. Наприклад, при висоті 4, ширині 7 та заповнювачі `*` результат у файлі потрібен бути наступним:

```
   *
  * *
 *   *
* * * * * * *
```

3. (`**`) Розробити програму, що в заданому файлі створює горизонтальну синусоїду із заданих символів (наприклад, зірочок). Амплітуда синусоїди, символ заповнювача та період задається у вхідному файлу. Наприклад, при амплітуді 5, періоді 40 та заповнювачі `*` результат у файлі потрібен бути наступним:

```
        *****
      ***   ***
     *         *
   **           **
  *               *
 *                 *
                     *               *
                       **           **
                         *         *
                           ***   ***
                              *****
```

Серія завдання **“про матриці”**:

Дано файл, що містить інформацію про матрицю. В першої строці надається інформація про розмір (напр., “*3 5*”). У кожному наступному рядку надається інформація про елементи рядку матриці. Результат операції записати до результуючого файлу.

4. Визначити максимальний елемент
5. Визначити суму елементів, що розташовані поверх головної діагоналі.
6. Визначити суму елементів, що розташовані знизу побічної діагоналі.
7. Визначити всі прості елементи діагоналі
8. (`*`) Визначити детермінант матриці якщо він існує.

Для виконання наступних завдань рекомендується використовувати функції та структури хедер файлу `dirent.h`:

9. (`*`) Видати на екран розмір запитаного файлу та його атрибути. Виклик функції *system* заборонено.
10. (`**`) Вивести структуру файлів та каталогів, як це робить утиліта Linux *tree*. Виклик функції *system* заборонено.
11. (`**`) Визначити об’єм запитаного каталогу. Результат нормалізувати. Формат розміру:
    - не більше ніж 3 знаки до коми;
    - не більше ніж 2 знаки після коми;
    - єдиний випадок, коли в чисельний частині числа може бути 0 - коли розмір дорівнює 0 байт.

Додаткові обов’язкові умови виконання робот

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- звіт повинен бути оформлений згідно “Вимогам до структурної побудови звіту”;
- продемонструвати відсутність витоків пам’яті за допомогою утиліти *valgrind*;
- доступ до елементів масиву здійснювати через розіменування покажчиків, а не через

оператор індексування (*[ ]*);

- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами.

Контрольні питання.

1. Що таке файл?
2. Які існують функції неформатованого введення даних?
3. Які існують функції неформатованого виведення даних?
4. Як визначити розмір файлу?
5. Як виконувати читання даних з файлу, коли кількість їх невідома?
6. Чи можна форматувати дані при їх записі у файл?
7. Як ввести з файлу та записати у файл рядки?
8. Як ввести з файлу та записати у файл символ?
9. Порівняйте текстові та бінарні файли.

Лабораторна робота №14. Структуровані типи даних

Загальне завдання

- з розділу “Індивідуальні завдання комплексної роботи” взяти прикладну галузь стосовно номеру варіанту за попередньо-визначеною формулою
- створити структуру, що відображає “базовий клас”

Структура директорії для лабораторної роботи

```
└── lab14
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── assets
    │   └── input.txt
    ├── doc
    │   └── lab14.md
    ├── test
    │   └── test.c
    └── src
        ├── lib.c
        ├── lib.h
        └── main.c
```

Зверніть увагу. Вхідні дані розташовані у файлі `assets/input.txt`. Програма повинна прийняти у якості аргументу шлях до цього файлу (як і у попередній роботі).

Зверніть увагу. Очікується, що при запуску програми з параметрами командної строки, нічого від користувача не буде очікуватись.

Зверніть увагу. Передача об’єктів структури в функцію (та отримання результатів роботи функції, що є об’єкт типу структури) обов’язково повинні передаватися “за вказівником”.

Індивідуальні завдання.

Обов’язкові завдання

- розробити функцію, яка читає дані (масив елементів) з файлу;
- розробити функцію, яка записує дані (масив елементів) у файл;
- розробити функцію, яка виводить масив елементів на екран;
- реалізувати функцію №1 з категорії “Методи для роботи з колекцією”, на вхід якої потрапляє масив об’єктів. Слід звернути увагу, що усі необхідні дані повинні бути передані як аргументи функції. Наприклад, якщо треба знайти всі машини марки “Форд”, то функція потрібна мати аргумент “марка машини”, та у main() викликати цю функцію з потрібним значенням марки.
- розробити функцію, яка буде сортувати масив елементів за заданим критерієм (полем);

Додаткові завдання на розсуд викладача

1. Розробити функцію, яка генерує елемент прикладної галузі згідно з індивідуальним завданням, при цьому:
    - чисельні поля генеруються за допомогою функції *rand()* у діапазоні *[N..N*10*], де* N* – номер варіанту;
    - рядкові поля подані у вигляді конкатенації двох рядків:, напр., “Студент №” та унікального числа.

2. Виконати запис та читання масиву структур (з заздалегідь відомою кількістю елементів) у двох форматів: текстовому та бінарному. При бінарному методі зберігання структур виконати пошук та читання структури з файлу по індексу (за допомогою використання функції *fseek*).

Додаткові обов'язкові умови виконання робот

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- звіт повинен бути оформлений згідно "Вимогам до структурної побудови звіту";
- продемонструвати відсутність витоків пам'яті за допомогою утиліти *valgrind*;
- доступ до елементів масиву здійснювати через розіменування покажчиків, а не через оператор індексування (*[ ]*);
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами.

Контрольні питання.

1. Як виконати доступ до окремих елементів структури?
2. Чи можна вміст однієї структури привласнити іншій того ж типу, використовуючи звичайний оператор присвоювання?
3. Коли необхідно використовувати покажчики на структури?
4. Чи може бути членом структури інша структура?
5. Чим відрізняється об'єднання від структури?
6. Чим відрізняються бітові поля від звичайних структур?
7. Що являє собою перерахування?
8. Що являє собою суміш?

Лабораторна робота №15. Динамічні масиви

Структура директорії для лабораторної роботи

```
└── lab15
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab15.md
    ├── test
    │   └── test.c
    └── src
        ├── entity.c
        ├── entity.h
        ├── list.c
        ├── list.h
        └── main.c
```

Загальне завдання:

На базі попередньо розробленого функціоналу по роботі з прикладною областю сформувати **динамічний масив** елементів розробленої структури. Реалізувати наступні функції роботи зі списком:

- вивід вмісту списку на екран;
- реалізувати функцію №1 з категорії “Методи для роботи з колекцією” (див. завдання з РЗ);
- додавання об’єкта у кінець списку;
- видалення об’єкта зі списку за індексом.
- сортування вмісту списку за одним з критеріїв

Додаткові обов’язкові умови виконання робот

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- звіт повинен бути оформлений згідно “Вимогам до структурної побудови звіту”;
- продемонструвати відсутність витоків пам’яті за допомогою утиліти *valgrind*;
- доступ до елементів масиву здійснювати через розіменування покажчиків, а не через оператор індексування (*[ ]*);
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами.

Контрольні питання.

1. Яким чином можна додавати нові елементи до динамічного масиву?
2. Опишіть структуру для створення динамічного масиву.
3. Чим відрізняється динамічний масив від звичайного?
4. Як виконати додавання елементу у динамічний масив?
5. Як виконати виділення елементу з динамічного масиву?

Лабораторна робота №16. (x2) Динамічні списки

Структура директорії для лабораторної роботи

```
└── lab16
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab16.md
    ├── test
    │   └── test.c
    └── src
        ├── entity.c
        ├── entity.h
        ├── list.c
        ├── list.h
        ├── menu.c
        ├── menu.h
        └── main.c
```

Основне завдання

На базі попередньо розробленого функціоналу по роботі з прикладною областю сформувати **односпрямований список** елементів розробленої структури. Реалізувати наступні функції роботи зі списком:

- читання даних з файлу, використовуючи функцію fscanf;
- запис даних у файл, використовуючи функцію fprintf;
- вивід вмісту списку на екран;
- реалізувати функцію №1 з категорії “Методи для роботи з колекцією” (див. завдання з РЗ);
- додавання об’єкта у кінець списку;
- видалення об’єкта зі списку за індексом.
- сортування вмісту списку за одним з критеріїв

А також:

- реалізувати діалоговий режим спілкування з користувачем за допомогою меню для демонстрації розроблених методів;
- продемонструвати відсутність витоків пам’яті;
- розробити модульні тести, що демонструють коректність роботи реалізованих функцій;

Додаткові завдання

- переробити метод додавання з можливістю вставлення додаткового елементу після будь-якого елементу списку;
- переробити список на використання двоспрямованого списку;
- реалізувати функцію обрахування, чи є список закільцьованим (зацикленим)

Додаткові обов’язкові умови виконання робот

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- звіт повинен бути оформлений згідно “Вимогам до структурної побудови звіту”;
- продемонструвати відсутність витоків пам’яті за допомогою утиліти *valgrind*;

- доступ до елементів масиву здійснювати через розіменування покажчиків, а не через оператор індексування (*[ ]*);
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами.

Контрольні питання.

1. Як виконати доступ до заданого за номером елементу списку?
2. Яким чином можна додавати нові елементи до списку?
3. Яке призначення головного елементу списку?
4. Як визначається кінець списку?
5. Скільки покажчиків може мати елемент списку?
6. Опишіть структуру для створення односпрямованого списку.
7. Що таке список?
8. Навіщо потрібний головний елемент у списку?
9. Як поміняти місцями два елементи, адреси яких задані?
10. Перерахуйте усі переваги і недоліки списку у порівнянні з масивом?
11. Опишіть структуру для створення двоспрямованого списку.
12. Чим характеризується двоспрямований список?
13. Навіщо потрібні головний та хвостовий елементи у списку?
14. Напишіть програмний код для переміщення від голови до хвоста списку.
15. Напишіть програмний код для переміщення від хвоста до голови списку.
16. Скільки покажчиків може мати елемент списку?
17. Чим відрізняється лінійний список від нелінійного?
18. Що таке “кільцевий двоспрямований список”? Як його створити?
19. Чому при сортуванні вмісту списку виконується не переміщення вмісту поля даних, а змінюється порядок елементів у списку шляхом зміни покажчиків на наступні елементи?
20. Скільки покажчиків може мати елемент списку? Що таке “мультисписок”?

Модуль 6. Об’єктно-орієнтовне програмування

Лабораторна робота №17. ООП. Вступ до ООП

Prerequisites

- Створити *новий* репозиторій (по аналогії з інснуючим) з назвою: *programing-FNAME-cpp* (`programing-davydov-cpp`):
- Додати до репозиторію файли:
  - `.clang-format` з попереднього репозиторію
  - `.clang-tidy` з попереднього репозиторію
  - `.gitlab-ci.yml` з наступним контентом:

```
include: https://gitlab.com/davs/cicd-pro-checker/-/raw/master/subjects/pro2/.gitlab-ci.yml
```

- модульні тести потрібно (пере)робити за допомогою Google Test Framework: https://google.github.io/googletest/

Структура директорії для лабораторної роботи

```
└── lab-cpp-01
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab-cpp-01.md
    ├── test
    │   └── test.cpp
    └── src
        ├── entity.cpp
        ├── entity.h
        ├── list.cpp
        ├── list.h
        └── main.cpp
```

Загальне завдання.

Для предметної галузі з розділу “Розрахункове завдання / Індивідуальні завдання” розробити два класи:

- клас, що відображає сутність “базового класу”, у тому числі:
  - конструктор за замовчуванням, копіювання та конструктор з аргументами (реалізація конструкторів повинна бути продемонстрована за допомогою списків ініціалізацій);
  - деструктор;
  - гетери та сетери на поля класу;
  - метод виводу об’єкта на екран:

    ```
    void Phone::print();
    ```

- клас, що має у собі динамічний масив об’єктів базового класу та має в собі методи додавання, видалення елемента, отримання елемента по індексу (або ідентифікатору), вивід усіх елементів на екран. Рекомендовані сигнатури методів:
  - додавання:

    ```
    void List::addPhone(const Phone& phone, size_t pos = 0);
    ```

  - видалення:

    ```
    void List::removePhone(size_t index);
    ```

– отримання по індексу:

```
Phone& List::getPhone(size_t index);
```

– виведення усіх елементів:

```
void List::print();
```

– метод 1 обходу колекції. Приклад сигнатури такого методу (У наведеному прикладі реалізоване завдання пошуку самого дешевого телефону з заданою діагоналлю (повертається один телефон):

```
const Phone& List::findCheapestPhone(unsigned float diagonal);
```

Додаткові умови виконання завдання

- усі поля “базового класу” повинні бути приватними та мати публічні гетери та сетери (модифікатори доступу), використовувати механізм інкапсуляції;
- усі функції, що не повинні змінювати поля поточного об’єкта, повинні бути константними;
- усі аргументи функцій, що не змінюються, по можливості також повинні бути константними. Якщо їх не можна зробити константними, у такому разі повинно бути обґрунтування цього;
- у класі-списку метод додавання елемента не повинен вводити дані з клавіатури або файлу, а повинен приймати вже готовий об’єкт для додавання. Метод вводу даних має бути відокремленим;
- продемонструвати відсутність витоків пам’яті;
- продемонструвати роботу розроблених методів класу-списку за допомогою модульних тестів.
- конструктори та деструктори повинні мати логіруючі повідомлення.

Студент повинен продемонструвати виклик деструктора та кожного типу конструктора, а також пояснити, коли вони викликаються;

Контрольні запитання

1. Що таке клас? Чим він відрізняється від структури?
2. Що таке метод? Чим він відрізняється від функції?
3. Що таке інкапсуляція?
4. Що таке константні методи? Наведіть приклади.
5. Як визначити розмір об’єкта у пам’яті?
6. Для чого потрібні права доступу?
7. Які методи потрібні для доступу до private-атрибутів?
8. Для чого потрібні const-методи?
9. Коли є смисл у константних методах?
10. Чи може бути атрибут константним?
11. Для чого потрібні конструктори?
12. Які ви знаєте відмінності конструкторів?
13. Чи можна перевантажувати конструктори?
14. Для чого потрібні деструктори?
15. Які відмінності деструктора?
16. Що таке конструктор копіювання? Для чого він потрібен?
17. Коли викликаються конструктори та деструктор?
18. Що таке списки ініціалізації?
19. Що таке перевантаження методів?
20. Чим визначається виклик перевантажених методів?

Лабораторна робота №18. ООП. Потоки

Структура директорії для лабораторної роботи

```
└── lab-cpp-02
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab-cpp-02.md
    ├── test
    │   └── test.cpp
    └── src
        ├── entity.cpp
        ├── entity.h
        ├── list.cpp
        ├── list.h
        └── main.cpp
```

Загальне завдання.

Поширити попередню лабораторну роботу таким чином:

- використання функцій `printf`/`scanf` замінити на використання `cin`/`cout`;
- усі конкатенації рядків замінити на використання `stringstream`;
- замінити метод виводу інформації про об'єкт на метод, що повертає рядок-інформацію про об'єкт, який далі можна виводити на екран;

```
std::string& Phone::toString();
```

- замінити метод вводу інформації про об'єкт на метод, що приймає рядок з інформацією про об'єкт, обробляє його та створює об'єкт на базі цієї інформації
- поширити клас-список, шляхом реалізації методів роботи з файлами за допомогою файлових потоків (fstream) (якщо використовувалися функції `fprintf`/`fscanf` – замінити їх на класи `ifsteam`/`ofstream`), при цьому сигнатури методів повинні виглядати таким чином:
  - читання (List – клас-список об'єктів, при цьому слід пам'ятати, що при повторному читанні з файлу, попередні дані списку повинні бути очищені):

```
void List::readFromFile(std::string& fileName);
```

  - запис:

```
void List::writeToFile(std::string& fileName);
```

Додаткові обов'язкові умови виконання робот.

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- робота повинна бути оформлена згідно "Вимогам до структурної побудови звіту";
- продемонструвати відсутність витоків пам'яті за допомогою утиліти *valgrind*;
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами;
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- не використовувати конструкцію "using namespace std;", замість цього слід робити "using" кожного необхідного класу, наприклад: using std::string, using std::cout;
- у проекті не повинні використовуватися бібліотеки введення / виведення мови С, а також не повинні використовуватися рядки типу `char*`.

Контрольні запитання

1. Як здійснювати виведення даних на екран за допомогою потоків?
2. Як здійснювати читання даних з клавіатури за допомогою потоків?
3. Для чого потрібен клас stringstream?
4. Для чого потрібен клас string? Наведіть аналогію роботи з типом char*.
5. Що таке простір імен?
6. Як здійснювати виведення даних у текстовий файл за допомогою потоків?
7. Як здійснювати читання даних з файлу за допомогою потоків?
8. Як здійснювати виведення даних у бінарний файл за допомогою потоків?
9. Яке призначення маніпуляторів setw(w) та setprecision(d)? Що треба зробити, щоб можна було їх використовувати?
10. Порівняйте текстові та бінарні файли. Яка у них відмінність?

Лабораторна робота №19. ООП. Перевантаження операторів

Структура директорії для лабораторної роботи

```
└── lab-cpp-03
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab-cpp-03.md
    ├── test
    │   └── test.cpp
    └── src
        ├── entity.cpp
        ├── entity.h
        ├── list.cpp
        ├── list.h
        └── main.cpp
```

Загальне завдання.

Поширити попередню лабораторну роботу (потоковий і/о при роботі зі класами) таким чином:

- у базовому класі (прикладної галузі) перевантажити:
  - оператор присвоювання;
  - оператор порівняння (на вибір 2 протележних оператора: == , != ; < , >= ; > , <=);
  - оператори введення / виведення;
- у класі-списку перевантажити:
  - оператор індексування ([ ]);
  - оператори введення / виведення з акцентом роботи, у тому числі і з файлами

Додаткові обов'язкові умови виконання робот.

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- робота повинна бути оформлена згідно "Вимогам до структурної побудови звіту";
- продемонструвати відсутність витоків пам'яті за допомогою утиліти *valgrind*;
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами;
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- не використовувати конструкцію "using namespace std;", замість цього слід робити "using" кожного необхідного класу, наприклад: using std::string, using std::cout;
- у проекті не повинні використовуватися бібліотеки введення / виведення мови С, а також не повинні використовуватися рядки типу `char*`.

Контрольні запитання

1. Для чого потрібні оператори?
2. Які оператори можна перевантажувати?
3. Які оператори не можна перевантажувати?
4. Що таке серіалізація та десеріалізація?
5. Як можна перевантажити оператори?

6. Чим відрізняється перевантаження операторів за допомогою функцій-членів класу та дружніх функцій?
7. Що таке “дружні функції”, у чому їх особливості, коли вони необхідні?

Лабораторна робота №20. (x2) ООП. Спадкування. Поліморфізм

Структура директорії для лабораторної роботи

```
└── lab-cpp-04
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   ├── lab-cpp-04.md
    │   └── assets
    │       └── uml.png
    ├── test
    │   └── test.cpp
    └── src
        ├── entity.cpp
        ├── entity.h
        ├── list.cpp
        ├── list.h
        └── main.cpp
```

Загальне завдання

Модернізувати попередню лабораторну роботу (оператори) наступним чином:

- базовий клас зробити абстрактним. Додати абстрактні методи;
- додати класи-спадкоємці з розділу “Розрахункове завдання / Індивідуальні завдання”, котрі будуть поширювати функціонал “базового класу” відповідно до індивідуального завдання;
- повинно бути реалізовано усі 3 методи з розділу “Розрахункове завдання / Індивідуальні завдання”;
- клас-список переробити таким чином, щоб він міг працювати як з базовим класом, так і з його спадкоємцями. При цьому серед полів класу-списку повинен бути лише один масив, що містить усі типи класів ієрархії. Оновити методи, що працюють з цим масивом.
- у функціях базового класу та класів-спадкоємців обов’язкове використання ключових слів final та override.
- упевнитися, що оператори вводу/виводу працюють коректно для усіх класів їєрархії.
- у звіти необхідно навести UML діаграму ієрархії класів

Додаткові обов’язкові умови виконання робот.

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- робота повинна бути оформлена згідно “Вимогам до структурної побудови звіту”;
- продемонструвати відсутність витоків пам’яті за допомогою утиліти *valgrind*;
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами;
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- не використовувати конструкцію “using namespace std;”, замість цього слід робити “using” кожного необхідного класу, наприклад: using std::string, using std::cout;
- у проекті не повинні використовуватися бібліотеки введення / виведення мови С, а також не повинні використовуватися рядки типу `char*`.

Контрольні запитання

1. Для чого потрібне спадкування?
2. Як впливають права доступу атрибутів на спадкування?
3. Які бувають атрибути при спадкуванні і на що вони впливають?
4. Коли працює спадкування для об’єктів-нащадків?
5. Що таке “ієрархія” класів?
6. Які ієрархії у спадкуванні можуть бути? Наведіть приклади.
7. Чим відрізняється спадкування від агрегації?
8. Що таке поліморфізм?
9. Для чого потрібні віртуальні методи?
10. Які методи називають “чисто віртуальними”?
11. Що таке таблиця віртуальних методів?
12. Що таке абстрактний клас?
13. Чи можна створити об’єкт абстрактного класу?
14. Що таке інтерфейс?
15. Які умови необхідно виконати для реалізації поліморфізму?
16. Чим відрізняється абстрактний клас від інтерфейсу?
17. Чим відрізняється абстрактний клас від звичайного класу?
18. Яких правил треба дотримуватись при перевизначенні віртуальних методів?
19. Яке призначення ключових слів final та override?

Лабораторна робота №21. ООП. Шаблонні функції та класи

Структура директорії для лабораторної роботи

```
└── lab-cpp-05
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab-cpp-05.md
    ├── test
    │   └── test.cpp
    └── src
        ├── list.hpp
        └── main.cpp
```

Загальне завдання. (Задача не пов'язана з попередніми роботами).

- Зробити шаблоний клас-список (на базі динамічного масиву), що має шаблоноване поле масиву (для будь-якого існуючого типу даних)
- Створити наступні методи:
  - вивод вмісту масиву на екран;
  - визначити індекс переданого елемента в заданому масиві;
  - відсортувати елементи масиву;
  - визначити значення мінімального елемента масиву;
  - додати елемент до кінця масиву;
  - видалити елемент з масиву за індексом.

Додаткові обов'язкові умови виконання робот.

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- робота повинна бути оформлена згідно "Вимогам до структурної побудови звіту";
- продемонструвати відсутність витоків пам'яті за допомогою утиліти *valgrind*;
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами;
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- не використовувати конструкцію "using namespace std;", замість цього слід робити "using" кожного необхідного класу, наприклад: using std::string, using std::cout;
- у проекті не повинні використовуватися бібліотеки введення / виведення мови С, а також не повинні використовуватися рядки типу `char*`.

Контрольні запитання

1. Для чого потрібні шаблони?
2. Як описати шаблонну функцію?
3. Як використовувати шаблонну функцію?
4. Чим відрізняються шаблонні функції від звичайних функцій?
5. Які дії виконує компілятор при виклику шаблонної функції?
6. Що треба зробити, щоб як аргумент шаблонної функції можна було вказувати змінну класу?
7. Які використовуються ключові слова для оголошення типу шаблонного аргументу?

8. Для чого потрібні шаблонні класи?
9. Як декларувати шаблонні класи?
10. Що таке статичний поліморфізм?
11. Що таке спеціалізація?
12. Чи є шаблонними методи у шаблонному класі?

Лабораторна робота №22. (x2) ООП. STL. Вступ до Стандартної бібліотеки шаблонів

Структура директорії для лабораторної роботи

```
└── lab-cpp-06
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab-cpp-06.md
    ├── test
    │   └── test.cpp
    └── src
        ├── entity.hpp
        ├── list.hpp
        └── main.cpp
```

Загальне завдання

Поширити реалізацію лабораторної роботи “Поліморфірзм” наступним шляхом:

- замінити масив та CRUD (create/read/update/delete) методи роботи з ним на використання STL
- для предметної галузі з розділу “Розрахункове завдання / Індивідуальні завдання” реализувати/оновити всі методи роботи з колекцією на використання функцій STL
- додати функцію сортування коллекції з використанням функтора
- додати функцію об’єднання двох класів-списків

Додаткове завдання (на розсуд викладача)

1. Додати можливість об’єднання двох STL-контейнерів типу `map` (в файлі `list.hpp`).

```
map<string, list<string>>& mergeMaps(map<string, list<string>> &map1, map<string, list<string>> &map2);
```

При цьому, якщо в обох контейнерах існують однакові ключі, то значення повинні конкатенуватися, наприклад, якщо є дві мапи для країн:

- Мапа1:
  - Україна : Харків, Київ;
  - Росія: Москва, Бєлгород;
  - Білорусь: Мінськ, Бобруйськ.
- Мапа2:
  - Польща: Варшава;
  - Росія: Санкт-Петербург;
  - Україна: Харків, Запоріжжя;

то об’єднана мапа повинна містити таке:

- Україна: Харків, Київ, Запоріжжя;
- Росія: Москва, Бєлгород, Санкт-Петербург;
- Білорусь: Мінськ, Бобруйськ;
- Польща: Варшава.

2. Зробити клас-список (з основного завдання) STL ітеративним. Для демонстрації ітеративності, наступний код повинен працювати (Object - тип поточного базового класу):

```
MyList<Object> list;
for (Object &o : list) {
```

```
        // actions with object
    }
```

Додаткові обов'язкові умови виконання робот.

- програма має мати документацію, що оформлена за допомогою утиліти doxygen;
- робота повинна бути оформлена згідно "Вимогам до структурної побудови звіту";
- продемонструвати відсутність витоків пам'яті за допомогою утиліти *valgrind*;
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- у звіті навести ступень покриття коду модульними тестами. 50% - є мінімально допустимим відсотком покриття коду тестами;
- продемонструвати роботу розроблених методів за допомогою модульних тестів;
- не використовувати конструкцію "using namespace std;", замість цього слід робити "using" кожного необхідного класу, наприклад: using std::string, using std::cout;
- у проекті не повинні використовуватися бібліотеки введення / виведення мови С, а також не повинні використовуватися рядки типу `char*`.

Контрольні запитання

1. Що таке стандартна бібліотека шаблонів?
2. Які складові входять до STL?
3. Наведіть приклад використання циклу range-for.
4. Яка різниця між послідовними контейнерами vector та list?
5. Яка різниця між послідовними контейнерами vector та set?
6. Що таке "ітератор", яке його призначення?
7. Які бувають ітератори?
8. Які операції можуть застосовуватися до ітераторів?
9. Що таке алгоритми у стандартній бібліотеці шаблонів?
10. Які методи дозволяють об'єднувати колекції?
11. Які методи дозволяють виконувати сортування у колекції за заданим критерієм для вказаного діапазону?
12. Що таке функтор?
13. Як працює контейнер map?
14. Що таке алгоритми?
15. Що таке біндери? Яки бувають типи біндерів?
16. Що робить функція for_each ?
17. Які функції дозволяють виконувати пошук у контейнері?
18. Які функції дозволяють виконувати визначення кількості елементів у контейнері за заданими характеристиками?

Модуль 7. Додаткові лабораторні роботи

Лабораторна робота №x1. Макровизначення

Структура директорії для лабораторної роботи

```
└── lab-x1
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab-x1.md
    └── src
        ├── lib.c
        ├── lib.h
        └── main.c
```

Індивідуальне завдання

Виконати завдання з пулу завдань на свій розсуд. **Зверніть увагу**. Викладач має право надати вам додаткове завдання для виконання.

1. Сортування масиву з 100 тис. елементів “бульбашкою”
2. Створити файл у тимчасовій директорії (див. `tmpfile()`) з 100 тис. випадкових чисел. Запис до файлу виконати з використанням буферу (поведінка за умовчуванням) та без використанням буферу (див. `setbuf`, зокрема `setbuf(stdout, NULL);`), при цьому:
    - переключення між режимами повинно бути за допомогою макровизначення `BUFFERED`

Загальне завдання

Зробити наступну “кастомізацію”: якщо визначене макровизначення `DEBUG`, то виконувати наступне:

- при запуску програми, вивести дату та час компіляції файлу;
- для кожної розробленої функції, при виконані виводити її ім’я (сигнатуру);
- при запуску програми, виводити ім’я поточного файлу (з повним шляхом до нього);
- при завершенні програми вивести загальний час роботи програми за допомогою функції `clock()` та типу `clock_t`.

А також:

- Макровизначення слід “встановлювати” за допомогою опції `-D` при компіляції.
- Схеми алгоритмів не потрібні для даної роботи

Контрольні питання.

1. Які бувають стадії роботи компілятора?
2. Що таке макрос?
3. Як визначити макрозмінну?
4. Яка макродиректива відповідає за значення поточного файлу?
5. За що відповідають наступні макровизначення `_TIME_`, `_DATE_`?
6. Яке призначення директиви `#define`?

Лабораторна робота №x2. Вступ до блок-схем алгоритмів

Загальне завдання

Для кожної розробленої функції, що були виконані у попередній роботі, слід зробити схему алгоритмів згідно з ДСТУ. Схеми алгоритмів повинні бути відокремленими файлами в форматі *.png*

При виконанні категорично рекомендується використовувати утиліту [05-03], яка доступна як онлайн-сервіс, так і є можливість завантажити її на комп'ютер та використовувати офлайн.

У ході виконання роботи, ви повинні отримати наступну структуру.

```
.
└── lab-x2
    ├── README.md
    └── doc
        └── assets
            ├── function-main.png
            ├── function-sort.png
            ├── ... (any other png, representing functions' schemas)
            └── lab-x2.drawio
```

Починаючи з цієї роботи звіт повинен мати Схеми алгоритмів функцій, які він розробив.

Контрольні питання.

1. Для чого потрібен схематичний опис?
2. Які існують основні компоненти схем алгоритмів?
3. Які існують форми запису схем алгоритмів циклів?
4. Які пакети та інструменти існують для створення схем алгоритмів?

Лабораторна робота №x3. Регулярні вирази

Структура директорії для лабораторної роботи

```
└── lab-x3
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab-x3.md
    ├── test
    │   └── test.c
    └── src
        ├── lib.c
        ├── lib.h
        └── main.c
```

Індивідуальне завдання

1. Визначити, чи є наведена строка номером транспортного засобу, що зареєстрований в Україні (напр. `AX2345IP`)
2. Визначити, чи є наведена строка дійсним числом у науковому форматі (напр. `1.25e+5`)
3. Визначити, чи є наведена строка математичним виразом, що містить лише цілі числа та оператори `+-*/`
4. Визначити, чи є наведена строка Мобільним номером України у міжнародному форматі з урахуванням існуючих на поточний момент операторів (напр. `+380(63)-123-45-67`)
5. Визначити, чи є наведена строка часом у 24-годинному форматі (напр. `16:05`)

Контрольні запитання

1. Що таке регулярні вирази? Для чого вони потрібні?
2. Наведіть основні класи, що використовуються для роботи з регулярними виразами.
3. Що таке квантифікація?
4. Як здійснювати перерахування при роботі з регулярними виразами?
5. Як здійснюється пошук за допомогою регулярних виразів?
6. Як здійснюється заміна за допомогою регулярних виразів?

Лабораторна робота №x4. Створення бібліотек

Структура директорії для лабораторної роботи

```
└── lab-x4
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── module-library
    │   ├── Makefile
    │   ├── src
    │   │   ├── lib.c
    │   │   └── lib.h
    │   └── test
    │       └── test.c
    └── module-main
        ├── Makefile
        └── src
            └── main.c
```

Основне завдання:

- на базі попередньо розробленого функціоналу по роботі з регулярними виразами, сформувати статичну бібліотеку, яка повинна включати в себе прототип, реалізацію та модульні тести розроблених методів. Таким чином, в проекті повинні бути 2 модуля:
  - бібліотека-модуль розробленого функціоналу (module-library)
  - основний модуль, що використовує розроблену бібліотеки (module-main)
- файли бібліотеки мають бути 100 % doxygen документовані;
- за допомогою утиліти `nm` дослідити перелік функцій, що має бібліотека;
- проект бібліотеки повинен мати модульні тести, але вони не мають бути включені до скомпільованої бібліотеки.

Додаткові необов'язкові завдання:

- переробити розроблену статичну бібліотеку на динамічну;

Додаткові вимоги виконання завдання:

- оформлення звіту для цієї роботи не потрібно.

Контрольні питання.

1. Що важче створювати: статичну чи динамічну бібліотеку?
2. Припустимо, що дві програми використовують бібліотеки: одна з них статичну, друга – динамічну. Яка з двох програм завантажиться швидше? Чому?
3. Припустимо, що дві програми використовують бібліотеки: одна з них статичну, друга – динамічну. Яка з двох програм (файлів, що виконуються) буде займати менше місця на диску? Чому?
4. Що важче використовувати: статичну чи динамічну бібліотеку?
5. Що важче підтримувати (виправляти баги): статичну чи динамічну бібліотеку?
6. Опишіть порядок створення статичної бібліотеки.
7. Опишіть порядок створення динамічної бібліотеки.

Лабораторна робота №x5. ООП. Обробка виключних ситуацій

Структура директорії для лабораторної роботи

```
└── lab-x5
    ├── Doxyfile
    ├── Makefile
    ├── README.md
    ├── doc
    │   └── lab-x5.md
    ├── test
    │   └── test.cpp
    └── src
        ├── lib.cpp
        ├── lib.h
        └── main.cpp
```

Загальне завдання

У файлі розміщена інформація про N масивів.

У першому рядку міститься інформація про кількість масивів, у кожній наступній – інформація про кількість елементів у кожному масиві та власне дані масиву.

Необхідно реалізувати програму, що виконує перераховані нижче дії, причому кожна з них в окремій функції, поки користувач не введе замість назви файлу рядок `\exit`

Дії, що має виконувати програма, такі:

- введення з клавіатури назви вхідного файлу з даними;
- читання даних з файлу;
- виконання індивідуального завдання;
- введення з клавіатури імені вихідного файлу;
- запис результату операції у файл;
- доступ до елемента за індексом слід винести в окрему функцію, що виконує перевірку на можливість виходу за межі масиву.

Зверніть увагу. Слід окремо звернути увагу, що при обробці виключення цикл не повинен переривatись.

Індивідуальні завдання

1. Визначити суму двох масивів. Результат операції – масив.
2. Перемножити два масиви. Результат операції – масив.
3. Підрахувати середнє значення елементів масиву. Результат операції – масив з середніх значень кожного із вхідних масивів.
4. Знайти у масиві елемент з максимальним значенням. Результат операції – масив з максимальних елементів кожного із вхідних масивів.
5. Знайти у масиві елемент з мінімальним значенням. Результат операції – масив з мінімальних елементів кожного із вхідних масивів.
6. Визначити кількість додатних елементів у масиві. Результат операції – масив з кількості додатних елементів у кожному із вхідних масивів.
7. Визначити кількість елементів, які належать діапазону `[a, b]` (значення a та b ввести з клавіатури). Результат операції – масив з кількості знайдених елементів у кожному із вхідних масивів.
8. Визначити кількість від'ємних елементів у масиві. Результат операції – масив з кількості від'ємних елементів у кожному із вхідних масивів.

9. Знайти у масиві номер першого елемента з максимальним значенням. Результат операції – масив з номерів максимальних елементів кожного із вхідних масивів.
10. Знайти у масиві кількість елементів з максимальним значенням. Результат операції – масив з кількостей максимальних елементів кожного із вхідних масивів.

Контрольні запитання

1. Від чого захищають макроси assert?
2. Що не варто поміщати всередину макросів assert?
3. Для чого потрібні різні класи для породження виключень?
4. Для чого потрібні специфікатори щодо виключних ситуацій?
5. У яких методах не рекомендується породжувати виключення?
6. У чому полягає “розмотування стека”?
7. Яке призначення конструкції try-catch?

Інше

Індивідуальні завдання комплексної роботи

01. Файл

- Поля базового класу:
  - Прихований файл (наприклад: так, ні)
  - Назва файлу (наприклад: “Лаба01”)
  - Розмір файлу, Кб (наприклад: 10, 123.56)
  - Доступи (https://www.tutorialspoint.com/unix/unix-file-permission.htm) (структура, що даватиме можливість вказати можливість читання, запису, виконання)
  - Формат файлу (один з переліку: txt, docx, pdf, mp3, avi, mp4, mkv, exe, bat, jar)
- Спадкоємець 1 - Відеофайл. Додаткові поля:
  - Роздільна здатність (https://animoto.com/blog/news/hd-video-creation-sharing) (один з переліку: 360, 480, 720, 1080)
  - Частота кадрів (https://uk.wikipedia.org/wiki/Частота_кадрів) (наприклад: 24, 72)
- Спадкоємець 2 - Файл зображення. Додаткові поля:
  - Розмір зображення (структура, що має значення кількості пікселей по ширині та по висоті)
  - Кількість точок на дюйм (https://en.wikipedia.org/wiki/Dots_per_inch) (наприклад: 200, 300)
- Методи для роботи з колекцією:
  1. Обрати з каталогу всі файли більше 50 кБ
  2. Відсортувати за назвою та обрати другий файл, що матиме всі доступи (rxw)
  3. Знайти зображення, що має найменшу кількість пікселів

02. Комерційна Угода

- Поля базового класу:
  - Чи відкрита (наприклад: так, ні)
  - Назва (наприклад: “Мобільний додаток для магазину одягу”)
  - Сума, грн (наприклад: 10000, 25000)
  - Контакт замовника (структура, що містить ім’я, електронну пошту та з якої країни контакт)
  - Етап перемовин (один з переліку: отримано запит на реалізацію проекту, обговорення проекту, виконання оцінки проекту, узгодження деталей)
- Спадкоємець 1 - Афіліативна угода (угода про передачу потенційного замовника іншому виконавцю (афіліату) з умовою отримання відсотку від доходу у разі укладання угоди афіліату з клієнтом). Додаткові поля:
  - Контакт-афіліат (структура, що містить ім’я, електронну пошту, та з якої країни контакт)
  - Відсоток афіліативного прибутку (технічно може бути від 0 до 100, на практиці складає від 3 до 10)
- Спадкоємець 2 - Аутстаф угода (https://medium.com/@idealogic_company/outsourcing-vs-outstaffing-pros-and-cons-7a2066b04392). Додаткові поля:
  - Ім’я розробника (наприклад: “Олексій Петров”, “Коваль Сергій”)
  - Рівень розробника (один з переліку: junior, middle, senior)
  - Кількість годин на тиждень (наприклад: 20, 30, але не більше 40)
- Методи для роботи з колекцією:
  1. Обрати угоди на стадії “Виконання оцінки проекту” та “Узгодження деталей”

2. Для афіліативних угод - Знайти угоди з найбільшим відсотком прибутку серед угод з афіліатом в Україні
3. Відсортувати угоди за іменем контакту та обрати першу угоду з сумою більше 20000

03. Смичковий інструмент

- Поля базового класу:
  - Чи акустичний інструмент (наприклад: так, ні)
  - Фірма/Майстер (https://consordini.com/best-violin-brands/) (наприклад: Stentor, Cecilio, Yamaha)
  - Рік створення (наприклад: 1970, 1850)
  - Смичок (структура, що містить вагу смичка у грамах та матеріал деревка, один з переліку: бразильське дерево, пернамбуку, скловолокно)
  - Розмір (https://kennedyviolins.com/pages/violin-size-chart) (один з переліку: 1, ½ , ¼, ¾)
- Спадкоємець 1 - Скрипка. Додаткові поля:
  - Наявність містка (наприклад: так, ні)
  - Наявність підборідника (наприклад: так, ні)
  - Тип інструменту (один з переліку: сольний, оркестровий, універсальний)
- Спадкоємець 2 - Контрабас. Додаткові поля:
  - Наявність додаткової струни (наприклад: так, ні)
  - Довжина шпилю, см (https://en.wikipedia.org/wiki/Endpin) (наприклад: 45, 60)
- Методи для роботи з колекцією:
  1. Знайти в колекції найстарішу скрипку фірми Yamaha
  2. Обрати з колекції акустичні інструменти розміру ½
  3. Відсортувати колекцію за вагою смичка від найважчого до найлегшого

04. Гітара

- Поля базового класу:
  - Наявність звукознімача (наприклад: так, ні)
  - Фірма/Майстер (https://musiccritic.com/equipment/guitars/best-guitar-brands/) (наприклад: Fender, Yamaha)
  - Кількість ладів (наприклад: 25, 20, може бути від 19 до 27)
  - Струни (структура, що містить товщину струн, що визначається найтоншою струною та складає від 8 до 12 мм, та матеріалом, одним з переліку: синтетичні, нержавіюча сталь, золото, сплав сталі та нікелю, бронза, мідь)
  - Розмір (https://www.martinguitar.com/features-materials/sizetype/) (один з переліку: 1, ½ , ¼, ¾)
- Спадкоємець 1 - Акустична гітара. Додаткові поля:
  - Відстань між деками, мм (наприклад: 65, 80, 100)
  - Кількість струн (один з переліку: 6, 7, 12)
- Спадкоємець 2 - Електрогітара. Додаткові поля:
  - Кількість регуляторів (наприклад: 1, 2, 3)
  - Форма корпусу (https://muzline.ua/articles/kakiye-formy-korpusa-byvayut-u-gitar/) (один з переліку: Stratocaster, Super Strat, Telecaster, Les Paul)
- Методи для роботи з колекцією:
  1. Обрати інструменти з товщиною струн не менше 9 мм та без звукознімача
  2. Знайти електрогітару фірми Fender з найбільшою кількістю ладів
  3. Відсортувати колекцію за фірмою у зворотному алфавітному порядку та обрати другу гітару з бронзовими струнами

05. Вакансія

- Поля базового класу:
  - Наявність оплачуваної відпустки (наприклад: так, ні)
  - Позиція (один з переліку: Junior Java Developer, Middle Java Developer, Senior Android Developer, Sales Manager, Project Manager)
  - Назва компанії (наприклад: EPAM, SoftServe)
  - Кількість років досвіду (наприклад: 2, 4, 6)
  - Діапазон заробітної плати (структура, що містить нижню та верхню межі заробітної плати для даної позиції)
  - Необхідний рівень знання англійської (один з переліку: pre-intermediate, intermediate, upper-intermediate, advanced)
- Спадкоємець 1 - Вакансія на роботу в офісі. Додаткові поля:
  - Адреса офісу (наприклад: “Харків, Сумська, 1”)
  - Наявність спортзалу (наприклад: так, ні)
- Спадкоємець 2 - Віддалена вакансія. Додаткові поля:
  - Бажаний часовий пояс (один з переліку: GMT+2, PT, EST)
  - Наявність досвіду віддаленої роботи (наприклад: так, ні)
- Методи для роботи з колекцією:
  1. Обрати всі вакансії, що потребують більше 2 років досвіду та пропонують оплачувану відпустку
  2. Відсортувати вакансії за назвою компанії та обрати першу вакансію, що потребує англійської рівня Intermediate, або вище
  3. Знайти віддалену вакансію з найбільшою можливою заробітною платою

06. Десерт

- Поля базового класу:
  - Чи є десерт безглюкозним (наприклад: так, ні)
  - Назва десерту (наприклад: Фреш, Топік, Реннарді, Апельсинове Муале)
  - Маса порції, г (наприклад: 120, 500, 300)
  - Поживна цінність (структура, що містить кількість кКал, масу білків, жирів та вуглеводів на 100 г продукту)
  - Смак (один з переліку: кисло-солодкий, пряно-шоколадний, солодкий, гірко-солодкий, вершковий, кавовий)
- Спадкоємець 1 - Випічка. Додаткові поля:
  - Час випікання, хв (наприклад: 30, 45, 80)
  - Чи містить муку (наприклад: так, ні)
- Спадкоємець 2 - Десерт на основі желе. Додаткові поля:
  - Желююча речовина (один з переліку: желатин, пектин, агар-агар)
  - Маса желюючої речовини, г (наприклад: 10, 20)
- Методи для роботи з колекцією:
  1. Обрати десерт з найбільшим вмістом вуглеводів серед безглюкозних десертів
  2. Знайти найкалорійніший десерт з кисло-солодким смаком
  3. Знайти випічку з найменшим часом випікання та без муки

07. Співробітник

- Поля базового класу:
  - Наявність страхування (наприклад: так, ні)
  - Назва компанії (наприклад: “Epam”, “NIX Solution”, “Google”)

  - Досвід роботи, років (наприклад: 0, 2, 10)
  - Контактні дані (структура, що містить прізвище, ім'я та електронну пошту співробітника)
  - Характеристики (декілька з переліку: впевнений, стресо-стійкий, цілеспрямований, організований)
- Спадкоємець 1 - Програміст. Додаткові поля:
  - Рівень розробника (один з переліку: junior, middle, senior)
  - Мова програмування (один з переліку: Java, C++, Python)
- Спадкоємець 2 - Перекладач. Додаткові поля:
  - Мова перекладу (один з переліку: англійська, німецька, українська)
  - Наявність навички синхронного перекладу (наприклад: так, ні)
- Методи для роботи з колекцією:
  1. Визначити кількість співробітників компанії "Google" без наявності страхування
  2. Знайти перекладачів з навичкою синхронного перекладу та досвідом роботи не менше 5 років
  3. Знайти програмістів з рівнем Middle та досвідом роботи більше року

08. Навчальний заклад

- Поля базового класу:
  - Чи є навчання безкоштовним (наприклад: так, ні)
  - Назва закладу (наприклад: Берізка, ім. Каразіна)
  - Загальна кількість учнів/студентів (наприклад: 250, 6000)
  - Голова закладу (структура, що містить прізвище, ім'я та електронну пошту керуючого закладом)
  - Початок навчального дня (один з переліку: стандарт 8:00, пізніше стандарту 8:30, друга зміна 14:30)
- Спадкоємець 1 - Вищий навчальний заклад. Додаткові поля:
  - Рівень акредитації (один з переліку: I, II, III, IV)
  - Кількість підрозділів (наприклад: 7, 9)
- Спадкоємець 2 - Школа. Додаткові поля:
  - Тип (один з переліку: гімназія, ліцей, корекційна школа, спеціалізована школа)
  - Номер (наприклад: 75, 103, 124)
- Методи для роботи з колекцією:
  1. Знайти навчальний заклад з максимальною кількістю учнів/студентів
  2. Визначити кількість ВНЗ III рівня акредитації, у яких кількість студентів перевищує відмітку в 2000 людей
  3. Визначити школи, які надають безкоштовне навчання та відсортувати їх за назвою

09. Агентство

- Поля базового класу:
  - Наявність вихідних (наприклад: так, ні)
  - Назва агентства (наприклад: "Find your love", "OneLaw")
  - Кількість років на ринку послуг (наприклад: 1, 4, 10)
  - Голова агентства (структура, що містить прізвище, ім'я та електронну пошту керуючого агентством)
  - Місто надання послуг (один з переліку: Харків, Київ, Лондон)
- Спадкоємець 1 - Юридичне агентство. Додаткові поля:
  - Види послуг (один з переліку: консультація з розводів, захист в суді)
  - Кількість виграних справ у суді (наприклад: 7, 9)

- Спадкоємець 2 - Шлюбне агентство. Додаткові поля:
  - Спосіб надання послуг (один з переліку: організація листування, організація зустрічі)
  - Співпрацює з країнами, можливо декілька варіантів (наприклад: Україна, Польща, Румунія)
- Методи для роботи з колекцією:
  1. Знайти всі агентства, які працюють в місті Харків та мають стаж на ринку не менше 3 років
  2. Знайти юридичне агентство з послугою “Захист в суді” з найбільшою кількістю виграних справ
  3. Знайти шлюбні агентства які працюють без вихідних

10. Дисципліна в університеті

- Поля базового класу:
  - Чи проводиться кожного тижня (наприклад: так, ні)
  - Назва дисципліни (наприклад: музика, програмування, фізика)
  - Кількість годин дисципліни (наприклад: 1, 4, 10)
  - Викладач (структура, що містить прізвище, ім’я та електронну пошту викладача)
  - Місто проведення лекційних занять (один з переліку: НТУ “ХПІ” ВК 302, НТУ “ХПІ” ЕК 204, НТУ “ХПІ” ЕК 302, Каразіна 705)
- Спадкоємець 1 - Технічна дисципліна. Додаткові поля:
  - Наявність лабораторних робіт (наприклад: так, ні)
  - Кількість годин, відведених на практичні заняття (наприклад: 45, 150)
- Спадкоємець 2 - Гуманітарна дисципліна. Додаткові поля:
  - Професійний інвентар (один з переліку: мікрофон, мольберт, пюпітр, без інвентарю)
  - Кількість семінарних занять в курсі (наприклад: 6, 8)
- Методи для роботи з колекцією:
  1. Знайти дисципліни, які проводяться кожного тижня в Каразіна 705
  2. Знайти гуманітарну дисципліну, яка потребує пюпітр, а кількість годин дисципліни не менша ніж 100
  3. Знайти технічну дисципліну, в якій присутні ЛР, а місце проведення лекційних занять - НТУ “ХПІ”

11. Потяг

- Поля базового класу:
  - Чи потрібен капітальний ремонт (наприклад: так, ні)
  - Номер потягу (наприклад: AD1113D)
  - Кількість вагонів (наприклад: 10, 35)
  - Напрямок руху (структура, що містить початковий та кінцевий пункти маршруту потяга)
  - Тип потягу (один з переліку: тепловоз, паровоз, електропотяг)
- Спадкоємець 1 - Пасажирський потяг. Додаткові поля:
  - Ціна квитка, USD (наприклад: 10, 15, 20)
  - Кількість місць у вагоні (наприклад: 100, 150)
- Спадкоємець 2 - Вантажний потяг. Додаткові поля:
  - Тип вантажу (один з переліку: дерево, вугілля, хлор, нафта)
  - Маса вагону, т (наприклад: 20, 50)
- Методи для роботи з колекцією:
  1. Знайти всі потяги з кількістю вагонів більш ніж 10, які потребують капітального ремонту

2. Знайти всі вантажні потяги, що прямують з України
3. Знайти вантажний потяг з найбільшою масою, серед тих, що перевозять дерево

12. Аудиторія

- Поля базового класу:
  - Чи зайнята (наприклад: так, ні)
  - Назва (наприклад: 25а, 313)
  - Кількість місць (наприклад: 30, 120)
  - Розташування (структура, що містить корпус та поверх, де знаходиться аудиторія)
  - Стан (один з переліку: необхідний ремонт, необхідний косметичний ремонт, не потребує ремонту)
- Спадкоємець 1 - Лекційна аудиторія. Додаткові поля:
  - Наявність проектору (наприклад: так, ні)
  - Кількість дошок (наприклад: 1, 2)
- Спадкоємець 2 - Лабораторна аудиторія. Додаткові поля:
  - Чи є закріплений лаборант (наприклад: так, ні)
  - Кількість комп'ютерів (наприклад: 5, 10, 15)
- Методи для роботи з колекцією:
  1. Знайти аудиторії, що вміщують більше 150 осіб
  2. Знайти вільні лекційну аудиторію з проектором
  3. Знайти аудиторії в певному корпусі, наприклад, ВК

13. Годинник

- Поля базового класу:
  - Чи є водонепроникним (наприклад: так, ні)
  - Назва моделі (наприклад: EFR-526L-1AVUEF, CS 55)
  - Ціна, USD (наприклад: 300, 1200)
  - Виробник (структура, що містить назву фірми та країну її місцезнаходження)
  - Стиль (один з переліку: спорт, класика, фешн)
- Спадкоємець 1 - Механічний годинник. Додаткові поля:
  - Наявність автопідзаводу (наприклад: так, ні)
  - Наявність скелетону (https://en.wikipedia.org/wiki/Skeleton_watch) (наприклад: так, ні)
- Спадкоємець 2 - Кварцовий годинник. Додаткові поля:
  - Тип батареї (один з переліку: сонячна, звичайна)
  - Ємність батареї, mAh (наприклад: 250, 330)
- Методи для роботи з колекцією:
  1. Знайти годинники з ціною менше 400$
  2. Знайти всі швейцарські годинники зі скелетоном
  3. Знайти всі годинники стилю "Класика"

14. Взуття

- Поля базового класу:
  - Чи є ортопедичним (наприклад: так, ні)
  - Назва моделі (наприклад: Superstar, AirForce)
  - Ціна, USD (наприклад: 200, 220)
  - Розмір (структура, що містить розмір та довжину устілки)
  - Бренд (один з переліку: adidas, puma, reebok, nike)

- Спадкоємець 1 - Кросівки. Додаткові поля:
  - Чи є біговими (наприклад: так, ні)
  - Призначення (один з переліку: спорт, повсякденне)
- Спадкоємець 2 - Чоботи. Додаткові поля:
  - Сезон (один з переліку: зима, осінь, весна)
  - Наявність антиковзаючої підошви (наприклад: так, ні)
- Методи для роботи з колекцією:
  1. Знайти ортопедичне взуття брендів “nike” та “puma”
  2. Знайти найдешевші бігові кросівки
  3. Знайти чоботи з розміром більше 39

15. Книга

- Поля базового класу:
  - Чи є електронна версія (наприклад: так, ні)
  - Назва (наприклад: Пригоди Тома Сойера)
  - Кількість сторінок (наприклад: 330, 510)
  - Видавництво (структура, що містить назву фірми та версію видання)
  - Палітурка (один з переліку: тверда, м’яка)
- Спадкоємець 1 - Художня книга. Додаткові поля:
  - Напрям (один з переліку: відродження, модерн, постмодерн)
  - Жанр (один з переліку: роман, детектив, новела, повість)
- Спадкоємець 2 - Наукова книга. Додаткові поля:
  - Сфера (один з переліку: хімія, біологія, фізика, програмування)
  - Чи є сертифікованою (наприклад: так, ні)
- Методи для роботи з колекцією:
  1. Знайти всі книги видавництва “Ранок”
  2. Знайти детективи, що мають електронну версію
  3. Знайти книгу з найбільшою кількістю сторінок

16. Студент

- Поля базового класу:
  - Чи на бюджетній формі навчання (наприклад: так, ні)
  - Прізвище та ініціали (наприклад: Назаренко Б.Є., Приліпа А.О.)
  - Куратор, прізвище та ініціали (наприклад: Свиридов К.В.)
  - Рік вступу до ВНЗ (наприклад: 2015, 2017, 2020)
  - Група (структура, що містить назву факультету, наприклад, КІТ, КН, та код групи, наприклад, 120а)
  - Корпус кафедри (один з переліку: ВК, ГАК, У1, У2)
- Спадкоємець 1 - Студент магістратури. Додаткові поля:
  - Чи змінював ВНЗ (наприклад: так, ні)
  - Бали ЄВІ (наприклад: 200, 180, 156)
- Спадкоємець 2 - Студент бакалаврату. Додаткові поля:
  - Наявність додаткових балів за підготовчі курси (наприклад: так, ні)
  - Попередній навчальний заклад (один з переліку: школа, технікум, коледж)
- Методи для роботи з колекцією:
  1. Знайти студентів, що вступили у 2018 році
  2. Знайти студентів, що вчились у коледжі
  3. Знайти студентів магістратури, які навчались до цього в інших ВНЗ

17. Рюкзак

- Поля базового класу:
  - Наявність відділу для ноутбуку (наприклад: так, ні)
  - Колір (наприклад: синій, зелений)
  - Об’єм, літри (наприклад: 20, 25)
  - Фірма (структура, що містить назву фірми та країну її місце знаходження)
  - Призначення (один з переліку: міський, тактичний, туристичний)
- Спадкоємець 1 - Шкіряний рюкзак. Додаткові поля:
  - Наявність підкладки (наприклад: так, ні)
  - Тип шкіри (один з переліку: анілінова, велюр, замша)
- Спадкоємець 2 - Рюкзак з тканини. Додаткові поля:
  - Чи є водонепроникним (наприклад: так, ні)
  - Тканина (один з переліку: синтетика, бавовна, брезент)
- Методи для роботи з колекцією:
  1. Знайти німецький шкіряний міський рюкзак з велюру
  2. Знайти замшеві рюкзаки без підкладки
  3. Знайти рюкзак синього кольору з найбільшим об’ємом

18. Мобільний телефон

- Поля базового класу:
  - Чи є водостійким (наприклад: так, ні)
  - Чи є ударостійким (наприклад: так, ні)
  - Модель або повна назва (наприклад: Nokia 3310)
  - Кількість оперативної пам’яті, Мб (наприклад: 2048, 1024)
  - Розмір сховища, Мб (наприклад: 2048, 4096)
  - Розмір екрану (структура, що містить ширину та висоту екрану у пікселях)
  - Операційна система (один з переліку: Android, iOS, Symbian, Windows Phone)
- Спадкоємець 1 - Кнопковий телефон. Додаткові поля:
  - Чи є бабусяфоном (наприклад: так, ні)
  - Кількість кнопок (наприклад: 10, 12)
- Спадкоємець 2 - Складаний телефон (наприклад: Samsung Z Flip). Додаткові поля:
  - Чи складаний телефон навпіл (наприклад: так, ні)
  - Розмір екрану у складеному стані (структура, що містить ширину та висоту екрану у пікселях)
- Методи для роботи з колекцією:
  1. Розрахувати загальний об’єм оперативної пам’яті
  2. Знайти кнопкові телефони, що НЕ є бабусяфонами
  3. Знайти всі складані телефони

19. Пошта

- Поля базового класу:
  - Чи є поштове повідомлення чернеткою (наприклад: так, ні)
  - Тема поштового повідомлення (наприклад: Подання документів)
  - Текст поштового повідомлення
  - Відправник (структура, що містить ім’я та поштову адресу)
  - Отримувач (структура, що містить ім’я та поштову адресу)
  - Кодування повідомлення (один з переліку: UTF-8, UTF-16, CP-1251)
- Обов’язкові методи базового класу:

  - Формування повідомлення шляхом об’єднання відправника, отримувача, теми та тіла письма
  - Розрахунок розміру повідомлення (алгоритм розрахунку - на розсуд студента)
- Спадкоємець 1 - Секретна пошта. Додаткові дії:
  - Необхідно перевизначити метод “формування повідомлення”, у якому всі числа повідомлення замінити на * та видалити знаки пунктуації
- Спадкоємець 2 - Стиснута пошта. Додаткові поля:
  - Ступінь стиснення (наприклад: 0.6, 0.3, 1, але завжди в межах від 0 до 1)
- Методи для роботи з колекцією:
  1. Знайти повідомлення, написані окремим користувачем
  2. Знайти всі секретні повідомлення
  3. Підрахувати загальний розмір усіх повідомлень

20. Лампочка

- Поля базового класу:
  - Чи ввімкнена лампочка (наприклад: так, ні)
  - Чи перегоріла лампочка (наприклад: так, ні)
  - Виробник лампочки (наприклад: ТОВ Рога та Копита)
  - Кількість вмикань лампочки до перегоряння, зворотній лічильник (наприклад: 20, 250)
  - Кількість ватт, які лампочка споживає кожну годину (наприклад: 5, 10, 15)
  - Температура кольору світіння лампочки (наприклад: 1800, 6600)
  - Форма (один з переліку: Candle, Tubular, Globe, Pear, Ogive)
  - Тип цоколю (один з переліку: E14, E27, E40)
- Обов’язковий метод базового класу:
  - Вмикання лампочки (зменшує лічильник вмикань перегорання, при значенні лічильника 0 встановлює прапор перегорання)
- Спадкоємець 1 - Розумна лампочка. Додаткові поля:
  - Чи наявне бездротове керування (наприклад: так, ні)
  - Назва мікроконтролера, на базі якого виконано лампочку (один з переліку: STM32F103, ESP8266)
  - Колір світіння лампочки в HEX (наприклад: #118038, #AC125E)
- Спадкоємець 2 - Вічна лампочка. Додаткові дії:
  - Перевизначити метод включення лампочки, таким чином, що поля кількості вмикань до перегоряння та прапору перегорання не змінюються
- Методи для роботи з колекцією:
  1. Знайти перегорілі лампочки
  2. Обчислити сумарне споживання (Вт), не враховуючи лампочки, що перегоріли
  3. Обрати усі розумні лампочки

21. Боєць

- Поля базового класу:
  - Чи є людиною (наприклад: так, ні)
  - Чи є чоловіком (наприклад: так, ні)
  - Унікальна назва бійця (наприклад: Чак Норіс)
  - Кількість одиниць життєвої сили (наприклад: 999, 123, 5)
  - Сила удару (наприклад: 100, 50)
  - Підсилення бійця (один з переліку: відсутні, захист 10, захист 20, додаткове життя)
  - Клан бойових мистецтв (один з переліку: зелений, червоний, синій)

- Спадкоємець 1 - Берсерк. Додаткові поля:
  - Вірогідність отримати підвищення сили удару (наприклад: 0.5, 0.7, але в межах від 0 до 1)
- Спадкоємець 2 - Сумоїст. Додаткові поля:
  - Вага сумоїста у кг, кожні 50 кг дають +10 до сили удару (наприклад: 150, 200)
- Спадкоємець 2 - Сумоїст. Додаткові методи:
  - Додати метод, що дає можливість сумоїсти з'їдати суші та таким чином додавати +5 кг до своєї маси. Алгоритм виклику методу - на розсуд студента
- Методи для роботи з колекцією:

1. Визначити, чи є на татамі представники різних кланів бойових мистецтв
2. Обрати бійців, що не мають підсилень, та не є берсерками
3. Виконати симуляцію битви. Поділити бійців на пари та з випадковою вірогідністю дати змогу вдарити першим одному з бійців. Якщо кількість бійців не парна, то останній боєць не приймає участі у битві. Повернути статус, чи трапилась битва. Якщо на татамі бійці лише одного клану бойових мистецтв, тоді битва не виконується. Не забудьте видалити бійців, які вже переможені

22. Країна

Необхідно описати країни, які існували під час Другої Світової війни.

- Поля базового класу:
  - Чи активну участь бере країна у бойових діях на поточний момент (наприклад: так, ні)
  - Назва країни (наприклад: Польща, СРСР)
  - Площа територій, кв.км (наприклад: 45513,1; 96523,1)
  - Бойова міць (структура, що містить опис бойової міці країн, а саме кількість піхоти, повітряних та морських збройних сил)
  - Устрій в країні до війни (один з переліку: монархія, республіка, …)
  - Устрій в країні після війни (один з переліку: монархія, республіка, …)
- Спадкоємець 1 - Країни-агресори. Додаткові поля:
  - Голова держави (наприклад: Гітлер, Сталін)
  - Кількість перемог у битвах (наприклад: 2, 5, 10)
  - Кількість поразок у битвах (наприклад: 2, 5, 10)
- Спадкоємець 2 - Другорядні країни. Додаткові поля:
  - Чи підвержено територію країни бойовим діям (наприклад: так, ні)
  - Ставлення до війни (один з переліку: нейтралітет, підтримка Антигітлерівської коаліція, підтримка Країни Осі)
- Методи для роботи з колекцією:
  1. Знайти усі країни, в яких було змінено устрій після війни
  2. Знайти країну із наймасштабнішою армією
  3. Знайти найбільш успішного голову держави (за кількістю перемог/поразок)

23. Птахи

У відділі орнітології почався перепис усіх зареєстрованих птахів. Вчені збирають наступну інформацію щодо птахів:

- Поля базового класу:
  - Чи окольцьована птаха (наприклад: так, ні)
  - Назва виду (наприклад: журавель, гусак)
  - Вік птаха, місяців (наприклад: 2, 6, 8)

  - Тип домівки птаха (структура, що містить площу у кв.см, висоту у см домівки птаха, а також кількість годівниць та наявність гнізда)
  - Стать птаха (один з переліку: чоловіча, жіноча)
- Спадкоємець 1 - Перелітні птахи. Додаткові поля:
  - Місяць відльоту у вирій (один з переліку: січень, лютий, березень, … , грудень)
  - Місяць прильоту з вирію (один з переліку: січень, лютий, березень, … , грудень)
- Спадкоємець 2 - Екзотичні птахи. Додаткові поля:
  - Мінімальна комфортна для життя температура, градусів Цельсію (наприклад: -5, +10, +15)
  - Максимальна комфортна для життя температура, градусів Цельсію (наприклад: +5, +20, +40)
- Методи для роботи з колекцією:
  1. Знайти відсоткове відношення самок до самців у відділі
  2. Знайти середній вік усіх не окольцьованих птахів
  3. Знайти птаха із найдовшою зимівлею

24. Контрольна робота

- Поля базового класу:
  - Чи перевірено роботу (наприклад: так, ні)
  - Прізвище студента (наприклад: Коваленко, Петренко)
  - Максимальна кількість балів (наприклад: 20, 40, але не більше 100)
  - Структура роботи (структура, що містить кількість практичних завдань, теоретичних питань та розгорнутих завдань)
  - Предмет (один з переліку: Програмування, Алгоритми, Електроніка)
- Спадкоємець 1 - Модульний контроль. Додаткові поля:
  - Кількість тем, за якими відбувається контроль (наприклад: 2, 3, 5)
  - Додаткові бали за активність на лекціях (наприклад: 1, 3, 6)
- Спадкоємець 2 - Іспит. Додаткові поля:
  - Кількість спроб студента (наприклад: 2, 3, 5)
  - Прізвище викладача, що приймає іспит (наприклад: Коваленко, Петренко)
- Методи для роботи з колекцією:
  1. Знайти усі контрольні роботи, в яких існуючи секції однакові за кількістю запитань (наприклад, (1,1,1), (2,2,2))
  2. Знайти усі контрольні роботи за певним предметом
  3. Знайти усі прізвища студентів, які склали іспит з першого разу

25. Самостійна робота

- Поля базового класу:
  - Чи є розбиття на варіанти (наприклад: так, ні)
  - Тема (наприклад: функції, одновимірні масиви)
  - Кількість завдань (наприклад: 3, 5, 10)
  - Предмет (структура, що містить назву предмету та прізвище викладача)
  - Частота проведення (один з переліку: раз на тиждень, раз на модуль, раз на семестр)
- Спадкоємець 1 - “Летучка”. Додаткові поля:
  - Наявність тестових запитань (наприклад: так, ні)
  - Наявність практичних завдань (наприклад: так, ні)
- Спадкоємець 2 - Лабораторна робота. Додаткові поля:
  - Тип завдання (один з переліку: написання програми, розрахунки, створення схеми, проведення експерименту)

  - Чи можна виконати роботу вдома (для виконання не потрібні спеціальні пристрої, прибори чи інструменті, яких немає вдома) (наприклад: так, ні)
- Методи для роботи з колекцією:
  1. Знайти самостійні роботи з найбільшою частотою проведення
  2. Знайти лабораторну роботу з написання програми, в якій найбільша кількість завдань
  3. Знайти лабораторні роботи без індивідуального варіанту завдання

26. Роботи

- Поля базового класу:
  - Чи робот-трансформер (наприклад: так, ні)
  - Загальна кількість пам'яті в умовних X-байтах (наприклад: 0, 2, 10)
  - Характеристики (структура, що містить оцінку інтелекту та сили від 1 до 100, та додаткове вміння, наприклад "телепатію")
  - Країна-виготовник (одна з переліку: США, Японія, Німеччина)
- Спадкоємець 1 - Озброєний робот. Додаткові поля:
  - Кількість боїв (наприклад: 2, 5, 10)
  - Назва зброї (наприклад: Зброя1, Зброя2)
- Спадкоємець 2 - Робот-автомобіль. Додаткові поля:
  - Пробіг, км (наприклад: 500, 2000, 1200)
  - Чи є безпілотним (наприклад: так, ні)
- Методи для роботи з колекцією:
  1. Знайти робота з найбільшим об'ємом пам'яті, що має здібність до рентгеного зору
  2. Знайти показник сили в найрозумнішого робота (за показником інтелекту)
  3. Знайти, чи є мирним найдосвідченіший у битвах робот (за кількістю битв)

27. Випускна кваліфікаційна робота

- Поля базового класу:
  - Чи робота захищена та здана (наприклад: так, ні)
  - Тема (наприклад: "Методи виявлення пневмонії")
  - Об'єм роботи, кількість сторінок без урахування додатків (наприклад: 120, 315, 207)
  - Автор роботи (структура, що містить прізвище, ім'я та по батькові автора)
  - Наукова сфера діяльності (один з переліку: лінгвістика, медицина, біологія, комп'ютерна інженерія)
- Спадкоємець 1 - Докторська робота. Додаткові поля:
  - Наявність експерименту (наприклад: так, ні)
  - Індекс Гірша (https://en.wikipedia.org/wiki/H-index) (наприклад: 1, 3)
- Спадкоємець 2 - Бакалаврська робота. Додаткові поля:
  - Ступінь бакалавра (один з переліку: звичайна, з відзнакою)
  - Можливість використання роботи як основу для магістерської (наприклад: так, ні)
- Методи для роботи з колекцією:
  1. Знайти всі роботи у сфері медицини та відсортувати їх за об'ємом роботи
  2. Знайти всі докторські роботи з наявним експериментом та об'ємом не меншим за 150 ст.
  3. Відсортувати всі бакалаврські роботи за темою, знайти ті з них, які отримали ступінь "З відзнакою"

28. Пристрій, що запам'ятовує

- Поля базового класу:

  - Чи має властивість перезапису (наприклад: так, ні)
  - Виробник (наприклад: Samsung)
  - Об’єм пам’яті, байт (наприклад: 10000000)
  - Габарити (структура, що містить довжину, висоту та ширину пристрою у мм)
  - Тип енергозалежності (один з переліку: non volatile, volatile, static, dynamic)
- Спадкоємець 1 - SSD. Додаткові поля:
  - Інтерфейс підключення (один з переліку: SATA, USB, Thunderbolt, M2)
  - Кількість циклів перезапису (наприклад: 10000)
- Спадкоємець 2 - Оптичний диск. Додаткові поля:
  - Тип оптичного диску (один з переліку: BD, CD, DVD, GD-ROM, MOD)
  - Діаметр диску, см (наприклад: 12)
  - Чи є диск двостороннім (наприклад: так, ні)
- Методи для роботи з колекцією:
  1. Знайти загальний об’єм пам’яті за виробником
  2. Знайти всі двосторонні диски типу Blu-ray (BD), виробник у яких компанія Samsung з об’ємом пам’яті більше 10 ГБ
  3. Знайти SSD з найбільшою кількістю циклів перезапису, у якого інтерфейс підключення - M2

29. Монітор

- Поля базового класу:
  - Чи є ваго-кріплення (наприклад: так, ні)
  - Назва (наприклад: Philips)
  - Діагональ, дюймів (наприклад: 12)
  - Інтерфейси підключення (структура, що містить кількість підключень типів VGA, HDMI, DVI, DisplayPort)
  - Розширення монітору (структура, що містить висоту та ширину монітору в пікселях)
  - Тип матриці (один з переліку: IPS, TN, VA, MVA, QD)
- Спадкоємець 1 - Ігровий монітор. Додаткові поля:
  - Частота оновлення, Гц (наприклад: 144)
  - Час відгуку, мс (наприклад: 5)
  - Чи вигнутий екран (наприклад: так, ні)
- Спадкоємець 2 - Плазмовий монітор. Додаткові поля:
  - Індекс якості зображення (наприклад: 2500)
  - Мережеві інтерфейси (структура, що відображає, чи підключено монітор до Wi Fi та кількість портів Ethernet)
  - Чи підтримує Smart TV (наприклад: так, ні)
- Методи для роботи з колекцією:
  1. Знайти всі монітори з діагоналлю більше 30 дюймів
  2. Знайти всі ігрові монітори з матрицею TN, частотою оновлення 144Гц та з часом відгуку менше 7 мс.
  3. Знайти плазмовий монітор з найменшою діагоналлю та підтримкою Smart TV.

30. Виконуючий файл

- Поля базового класу:
  - Чи є прихованим (наприклад: так, ні)
  - Назва (наприклад: Qwerty)
  - Розмір, байт (наприклад: 1200)

  - Атрибути (структура, що описує, чи є файл архівним (A), системним (S), дозволеним тільки на читання (R))
  - Розширення файлу (один з переліку: exe, msi, com)
- Спадкоємець 1 - Скриптовий файл. Додаткові поля:
  - Shabang (наприклад: `#!/bin/bash`)
  - Чи потрібен інтерпретатор для даного скриптового файлу (наприклад: так, ні)
- Спадкоємець 2 - Файл-вірус. Додаткові поля:
  - Рівень небезпечності (один з переліку: low, medium, critical, nightmare)
  - Тип вірусу (один з переліку: resident, macro, boot, non-resident, email)
- Методи для роботи з колекцією:
  1. Знайти всі файли, які придатні тільки для читання (R)
  2. Знайти всі bash скрипти
  3. Видалити всі вірусні файли з розширенням .exe та рівнем небезпечності вище low

31 Комп'ютерна гра

- Поля базового класу:
  - Чи є гра free-to-play (наприклад: так, ні)
  - Назва (наприклад: Stellaris, Need for speed)
  - Розмір, Гб (наприклад: 25, 73)
  - Розробник (структура, що містить назву компанії та назву підрозділу)
  - Ігровий рушій (один з переліку: EAGL4, Clausewitz Engine)
- Спадкоємець 1 - Гра стратегія. Додаткові поля:
  - Наявність ігрової паузи (ігрова пауза мається на увазі не у вигляді виходу в меню, а у вигляді окремої функції, за допомогою якої можна зупинити ігровий процес і внести корективи в поведінку ігрових об'єктів) (наприклад: так, ні)
  - Вид стратегії (наприклад: економічна, тактична, покрокова)
- Спадкоємець 2 - Гра гонка. Додаткові поля:
  - Кількість доступного транспорту (наприклад: 20, 45)
  - Кількість треків (наприклад: 50, 35)
- Методи для роботи з колекцією:
  1. Знайти всі ігри написані з використанням Clausewitz Engine (Ігровий рушій) та мають розмір більше за 40 Гб
  2. Знайти гру стратегію, розмір якої не більше 30 Гб, а вид даної стратегії - тактична стратегія
  3. Знайти гру гонку, яка має не менше 30 видів доступного транспорту та не є безкоштовною