Распределение баллов

№	Название	Баллы
1	Решение системы уравнений	10
2	Перекодировка	20
3	Сортировка	15
4	Длинная арифметика	30
	Итог	75 (max 70)

Полезные ссылки

Записи и трансляции лекций (twitch)	https://www.twitch.tv/pavelsx_itmo
Записи и трансляции лекций (youtube) (ожидается плейлист)	
C/C++ лекция 2021.02.20	https://www.youtube.com/watch?v=DK4aLf3Ei5U
C/C++ лекция 2021.02.27	https://www.youtube.com/watch?v=NEHd2wrsAI8
C/C++ лекция 2021.04.03	https://www.youtube.com/watch?v=EAjlL589QWQ
C/C++ лекция 2021.04.24	https://www.youtube.com/watch?v=Xkktb2taUFQ
C/C++ лекция 2021.05.08	https://www.youtube.com/watch?v=KyibzuoRQSk
C/C++ лекция 2021.05.29	https://www.youtube.com/watch?v=q26QOdVvvSU
C/C++ | П.С. Скаков (стримы)	https://youtube.com/playlist?list=PLYRa06HUj-l6qYcBsrKJyfDyYA0VCEhKF
Презентация (тестирование)	
pdf	https://drive.google.com/file/d/1GtKk5ug9QpwIhjybl6ERs8x5r7k5Poi1/view?usp=sharing
pptx	https://drive.google.com/file/d/159PauuKTYNGGgERYJ-1uouqHLGv8t0mN/view?usp=sharing

Лабораторная работа 1
Решение системы уравнений

Цель работы: изучить особенности работы с числами с плавающей точкой и массивами в C.

Описание:

Программа должна находить решение системы линейных уравнений.

Входной файл в первой строке содержит одно число: размер системы уравнений N, после чего идут N строк по N+1 числу, которые соответствуют коэффициентам при N переменных и свободному члену. Гарантируется корректность входных данных. N - натуральное число.

Для хранения элементов матрицы использовать тип float.

Выходной файл должен содержать:

- найденные значения переменных в формате одно число на каждой строке если решение единственно (значения печатаются как %g);
- только фразу “many solutions” если решение не единственно;
- только фразу “no solution” при отсутствии решений.

Пример входных данных:

2

0.5 3 4

0 2 5

Пример выходных данных:

-7

2.5

Аргументы программе передаются через командную строку:

`lab1 <имя_входного_файла> <имя_выходного_файла>`

Программа должна:

1. быть написана на C;
2. не использовать внешние библиотеки;
3. всегда корректно освобождать память и закрывать файлы;
4. обрабатывать ошибки: файл не открылся, не удалось выделить память – выдавать сообщение об ошибке и корректно завершаться с ненулевым кодом возврата (при отсутствии ошибок – завершаться с нулевым);
5. не писать в консоль ничего лишнего, кроме сообщений об ошибках и краткой справки по использованию (при запуске с неправильным числом аргументов).

Лабораторная работа 2
Перекодировка

Цель работы: изучить особенности работы с массивами и строками в C.

Описание:

Программа должна выполнять перекодировку текстового файла.

Аргументы программе передаются через командную строку:

`lab2 <имя_входного_файла> <имя_выходного_файла> <выходная_кодировка>`

где **`<выходная_кодировка>`:**

0 – UTF-8 без BOM;

1 – UTF-8 с BOM;

2 – UTF-16 Little Endian;

3 – UTF-16 Big Endian;

4 – UTF-32 Little Endian;

5 – UTF-32 Big Endian.

Входная кодировка: любая из перечисленных, должна определяться программой автоматически.

Неправильные UTF-8 байты кодировать/декодировать как символы из диапазона 0xDC80..0xDCFF. UTF-8 -> другая кодировка -> UTF-8 должно точно возвращать исходный файл (плюс/минус BOM).

Программа должна:

1. быть написана на C;
2. не использовать внешние библиотеки;
3. всегда корректно освобождать память и закрывать файлы;
4. обрабатывать ошибки: файл не открылся, не удалось выделить память – выдавать сообщение об ошибке и корректно завершаться с ненулевым кодом возврата (при отсутствии ошибок – завершаться с нулевым);
5. не писать в консоль ничего лишнего, кроме сообщений об ошибках и краткой справки по использованию (при запуске с неправильным

числом аргументов).

Лабораторная работа 3
Сортировка

Цель работы: изучить особенности работы со строками, структурами и рекурсией в C.

Описание:

Программа должна сортировать строки (однобайтовая кодировка символов, простое регистрозависимое сравнение по коду символа) файла в формате:

<фамилия><пробел><имя><пробел><отчество><пробел><телефон>

где:

- <фамилия>, <имя>, <отчество> – строки не длиннее 20 символов, не содержат пробелов, табуляции, переводов строк или ноль-символов;
- <телефон> – целое неотрицательное число, меньшее 10^{11}.

В порядке фамилия->(если совпадает)имя->отчество->телефон.

Каждый элемент строки должен храниться отдельным полем структуры. Порядок сортировки должно быть возможно изменить (например, первым учитывать имя) незначительными изменениями в программе.

Аргументы программе передаются через командную строку:
lab3 <имя_входного_файла> <имя_выходного_файла>

Программа должна:

1. быть написана на C;
2. не использовать внешние библиотеки;
3. метод сортировки: самостоятельно реализованный QuickSort (детерминированный, не должен падать при любых входных данных);
4. всегда корректно освобождать память и закрывать файлы;

5. обрабатывать ошибки: файл не открылся, не удалось выделить память – выдавать сообщение об ошибке и корректно завершаться с ненулевым кодом возврата (при отсутствии ошибок – завершаться с нулевым);
6. не писать в консоль ничего лишнего, кроме сообщений об ошибках и краткой справки по использованию (при запуске с неправильным числом аргументов).

Файлы с исходным кодом должны располагаться в одном каталоге (лежать на dev).

Лабораторная работа 4
Длинная арифметика

Цель работы: изучить особенности работы с классами в C++.

Описание:

Стандарт C++17.

В программе должен быть реализован класс LN, позволяющий выполнять арифметические операции над целыми числами произвольной точности в десятичной системе. С его использованием должна проводиться работа с входными и выходными файлами.

Входной файл содержит выражение в форме обратной польской записи. Каждое число и знак операции (‘+’, ‘-’, ‘*’, ‘/’, ‘%’ – остаток от деления, ‘~’ – корень квадратный, ‘_’ – унарный минус, ‘<’, ‘<=’, ‘>’, ‘>=’, ‘==’, ‘!=’) располагается на отдельной строке. Каждая строка оканчивается символом новой строки.

Выходной файл должен содержать состояние стека на момент завершения работы программы. Каждое значение находится на новой строке, начиная с вершины, строка оканчивается символом новой строки.

Квадратный корень и деление округляют к 0. Результат при взятии корня из отрицательного числа или делении на ноль: NaN. Результат операций сравнения: 0 (для false) или 1 (для true). Результат арифметических действий и сравнения с NaN: в соответствии со стандартом IEEE-754. Число -0 должно быть полностью эквивалентно 0 (включая вывод).

Аргументы программе передаются через командную строку:
`lab4 <имя_входного_файла> <имя_выходного_файла>`

Нехватка памяти при операциях с классом должна обрабатываться через исключения C++.

Класс должен иметь конструкторы:

- конструктор копирования,
- конструктор перемещения,
- из long long со значением по умолчанию: 0,
- из строки символов C (const char *),
- из строки std::string_view.

Для класса должны быть реализованы операторы:

- арифметические: +, -, *, /, %, ~ (как корень квадратный), - (унарный);
- комбинация арифметических операций и присваивания;
- сравнения: <, <=, >, >=, ==, !=;
- присваивания;
- перемещения;
- преобразования типа в: long long (с генерацией исключения в случае, когда значение не умещается), bool (неравенство нулю);
- создания из литерала произвольной длины с суффиксом _ln (например, должно работать выражение: LN x; x = 123_ln;).

Реализовать вспомогательные функции/методы: сложение, вычитание, универсальное сравнение (возвращает -1, 0, 1), которые будут использоваться в функциях/методах оператор ‘+’, оператор ‘<’ и пр. Например, при реализации и оператора ‘+’, и оператора ‘-’ необходимо выполнять и сложение, и вычитание, в зависимости от знаков входных чисел.

Объявление класса необходимо поместить в заголовочный файл LN.h с include guards (защитой от повторного включения), реализация крупных методов (больше 1 строки) класса должна быть в LN.cpp. Код функции main располагается в main.cpp. Все файлы должны лежать рядом с main.cpp.

Допустимо загружать тесты в репозиторий. Они должны находиться в папке tests рядом с main.cpp.

Классы необходимо оформлять согласно требованиям к оформлению, которые были обновлены к этой работе (пункт 1.16 и дальше).

Рекомендации по решению:

1. В классе отдельно хранить знак числа и отдельно разряды числа, при этом разряды рекомендуется хранить массивом, начиная с младшего (digits[0] соответствует самой младшей цифре числа).

Использовать stl внутри класса LN запрещено.

До лекции 22.05 вы можете считать класс структурой, а операторы реализовывать как обычные методы с произвольными именами. Переделка такого кода в финальный очень простая.

Программа должна:

1. быть написана на C++;
2. не использовать внешние библиотеки;
3. всегда корректно освобождать память и закрывать файлы;
4. обрабатывать ошибки: файл не открылся, не удалось выделить память – выдавать сообщение об ошибке и корректно завершаться с ненулевым кодом возврата (при отсутствии ошибок – завершаться с нулевым);
5. не писать в консоль ничего лишнего, кроме сообщений об ошибках и краткой справки по использованию (при запуске с неправильным числом аргументов);
6. умножение, деление и квадратный корень должны работать с адекватной скоростью (то есть требуется алгоритм уровня “в столбик”, а не “умножение на n путём сложения n раз”).

Инструкция по отправке результатов работ

Работы выполняются с использованием системы контроля версий (**Github Classroom**). Для работы с системой требуется зарегестрироваться на сайте https://github.com/.

После регистрации на сервере необходимо присоединиться к classroom. Это можно сделать, перейдя по ссылке для первой лабораторной работы. В качестве имени пользователя (student) нужно выбрать свой логин ИСУ. Если вы не нашли себя, то нужно написать Виктории (@viktoriya_yve) и она добавит вас в список.

После того, как вы примите задание, то вам будет автоматически создан приватный репозиторий, куда нужно будет заливать свой код, которые соответствует требованиям к оформлению. Категорически не допускается добавление в проект не относящихся к нему файлов, а также построенных программ, промежуточных файлов и файлов, создаваемых средами разработки, например:

a. *.sln, *.vcxproj и т.д. от Visual Studio;
b. *.pro и т.д. пот Qt;
c. idea от Clion и пр.

То есть, в репозиторий можно отправлять только файлы *.c, *.h, *.cpp, *.hpp и .gitignore (что рекомендуется использовать для автоматического избавления от мусора), если дополнительно ничего не указано в условии задания. Допускается добавление Readme.md с описанием решения.

Изначально у вас будет только 1 ветка main. Первым делом необходимо создать ветку с названием **dev**, на которую вы будете заливать свой код на проверку. Названия коммитов должны быть адекватными, а не рандомным набором символов. Описание опционально, но если при исправлении багов в описании коммита проверяющий увидит, что именно вы изменили, то это будет только плюсом.

Сдача работы осуществляется путем отправки Pull (**PR**) или Merge Request с **dev** на **main** с названием ***Attempt 1***. Сроком представления считается дата отправки запроса. В **Reviewers** ставим Викторию (собственно там будет только 1 вариант). Описание PR опционально как и в случае с описанием коммитов. *Важно*: Merge или Close PR делает проверяющий. Все содержимое запроса принимается или отклоняется

целиком. Наличие конфликтов при слиянии, а также лишних файлов, приводит к отклонению запроса и работы.

По результату проверки вы увидите либо Closed PR и статус ***Changes requested*** или Merged со статусом ***Approved***.

Если вы видите по результату проверки ***Changes requested*** (и закрытый PR), то смотрим в комментарий и идем исправлять баги. Код по-прежнему надо коммитить на ветку **dev**. Как будете готовы отправлять работу на повторную проверку, то создаем новый PR с названием ***Attempt 2***, ставим reviewer и ждем результатов. Поскольку попыток сдать работу 3, то после PR Attempt 3 остальные попытки не принимаются и на защиту вы выходите с тем, что есть.

Если вы видите ***Approved*** с комментарием, что всё ок/допущен до защиты и пр, то делаем merge PR, если проверяющий этого вдруг не сделал, и готовимся к защите.

По желанию, вы можете создать ветку **test**, где помимо основного кода будут лежать ваши тесты.

На **защите** вас ждут вопросы по работе, а именно, про реализацию алгоритма и конструкций C/C++, который вы использовали при написании кода. На защиту можно выходить, если по вашему коду больше, чем 0 баллов, то есть хоть что-то набрано. После защиты баллы за код поднять нельзя, поэтому сначала, если в вашем коде были ошибки, нужно их исправить, дождаться повторных проверок и только потом выходить на защиту.

Защита, как и сдача работы, может быть завалена (очень нежелательное явление). Если такое происходит, то человек идет готовиться и через неделю пытается защитить работу снова. Если вторая попытка защиты проваливается, то считается, что человек не понимает, что он сдает = 0 баллов за работу, даже если работает код.

Важно: если будет замечен плагиат кода, то и того, кто скатал, и того, у кого скатали, постигнет кара: отрицательные баллы за работу без права пересдачи. В этом случае баллы могут быть понижены в любое время, даже в конце семестра.

Примечание по требованиям к оформлению: если кого-то идеологически не устраивают какие-либо пункты оформления исходников, то это можно обсудить с Викторией.

Требования к оформлению работ

1. Оформление исходных текстов

1. Исходные тексты программы должны содержать не более одной команды на одной строке.
2. Недопустимо писать всю функции в одну строку: заголовок функции всегда должен идти на отдельной строке или нескольких, если имеет большую длину.
3. Длина одной строки не должна превышать 120 символов в большинстве случаев (120+ допустимо) и 140 символов всегда.
4. Переменные определяются максимально близко к месту использования (то есть не описывать все переменные в начале функции). Они должны иметь минимально возможную область видимости (например, счётчик цикла, нужный только для цикла, должен создаваться в цикле, а не быть внешней переменной).
5. Если вы инициализируете переменную, то это следует делать одновременно с её созданием.
6. Все указатели, через которые данные не изменяются, должны быть правильно отмечены квалификатором const.
7. Аргументы функции, переданные по ссылке и не изменяющиеся внутри функции, должны быть также отмечены квалификатором const.
8. Единицей отступа является **1 tab или 4 пробела**.
9. Отступы должны отражать логическую структуру программы:
    a. команды, выполняемые внутри циклов, должны иметь отступ на 1 больший, чем заголовок цикла;
    b. команды, выполняемые внутри **if** и **else**, должны иметь отступ на 1 больший, чем строки с **if** и **else**;
    c. метки **case** внутри **switch** должны быть выровнены на тот же отступ, что **switch**;

d. команды внутри **switch** должны иметь на больший 1 отступ, чем оператор **switch**;

e. команды внутри фигурных скобок должны иметь отступ на 1 больший, чем уровень отступа скобок.

10. Открывающая фигурная скобка тела функции, структуры и подобного всегда переносится на другую строку.

```
1 void foo(void)
2 {
3       ...
4 }
```

11. Фигурные скобки должны быть расставлены единообразно. Допускаются 2 варианта:

    a. Открывающая фигурная скобка переносится на следующую строку и имеет тот же отступ, что и предыдущая строка, закрывающая скобка находится на отдельной строке, и ее отступ совпадает с открывающей. Команды внутри скобок имеют дополнительный отступ. Ключевое слово **else** должно находится на отдельной строке. Например:

```
if (condition)
{
      c = a + b;
}
else
{
      c = a - b;
}
```

    b. Открывающая фигурная скобка завершает строку с оператором, закрывающая скобка находится на отдельной строке, и ее отступ совпадает с отступом строки, содержащей открывающую скобку (так называемые «египетские скобки»). Команды внутри скобок имеют дополнительный отступ. Ключевое слово **else** должно находиться на строке вместе с закрывающей скобкой. Например:

```
if (condition) {
      c = a + b;
} else {
```

```
      c = a - b;
}
```

12. Ключевые слова **else** и **if** могут быть объединены на одной строке для уменьшения отступов. Такой стиль следует использовать в случае однотипных условий. Пример:

```
if (condition1) {
      c = a + b;
} else if (condition2) {
      c = a - b;
}
```

13. Допустимо следующее оформление имен функций:

    a. camelCase: функции и методы должны начинаться с маленькой буквы, дополнительные слова должны начинаться с большой буквы и идти вместе с предыдущим, например, draw() или getArea();

    b. snake_case: функции и методы должны начинаться с маленькой буквы, дополнительные слова должны начинаться также с маленькой буквы, кроме аббревиатур, и отделяться от предыдущих подчеркиванием.

14. В названиях должны использоваться только латиница, цифры и подчеркивания. Недопустимо создавать имена, начинающиеся с подчеркиваний.

15. Перенос части длинных команд на другую строку должен выполняться следующим образом:

    a. в начале следующей строки должен находиться перенесенный оператор (за исключением запятой);

    b. отступ должен быть больше на 1 единицу отступа;

    c. в случае переноса внутри сложного выражения с круглыми скобками, отступ должен отражать вложенность скобок, каждый уровень вложения добавляет 1 отступ на следующий строке.

16. Имена классов должны начинаться с большой буквы. Если имя состоит из нескольких слов, они идут подряд без символов подчеркивания, каждое слово начинается с большой буквы (CapsCase). Например, Queue или RoundedRectangle. Исключением из правил могут является функторы, имена которых могут быть в нижнем регистре. В этом случае отдельные слова в имени разделяются подчеркиваниями.

17. Имена классов должны представлять собой имена существительные с определениями и дополнениями. Исключения: имена функторов должны иметь в основе глагол, так как представляют собой действие.

18. Структуры (без конструкторов и деструкторов, а также без методов) допустимо называть в нижнем регистре и разделять слова подчеркиваниями. В этом случае имя должно иметь суффикс «_t».

19. Секции внутри класса должны быть упорядочены в порядке убывания интереса к ним со стороны читающего код: первой должна идти секция **public**, так как это интерфейс класса для клиентов; затем секция **protected**, поскольку она представляет собой интерфейс для наследников; завершает класс секция **private**, так как эта информация интересна только разработчику класса.

20. Внутри каждой секции информация должна быть упорядочена следующим образом:

    a. типы;

    b. поля;

    c. конструктор по умолчанию;

    d. конструкторы копирования и перемещения;

    e. все остальные конструкторы;

    f. деструктор;

    g. перегруженные операторы;

    h. методы.

21. Допустимо следующее оформление имен методов:
    a. camelCase: функции и методы должны начинаться с маленькой буквы, дополнительные слова должны начинаться с большой буквы и идти вместе с предыдущим, например, draw() или getArea();
    b. snake_case: функции и методы должны начинаться с маленькой буквы, дополнительные слова должны начинаться также с маленькой буквы, кроме аббревиатур, и отделяться от предыдущих подчеркиванием.
22. Имена полей класса должны иметь отличительный признак поля, в качестве которого следует использовать либо префикс «m_», либо завершающий символ подчеркивания (предпочтительнее).
23. Недопустимо создавать имена, начинающиеся с подчеркиваний.
24. Не допускается использование конструкций using namespace на верхнем уровне в заголовочных файлах.

Наиболее простым способом следовать этим правилам будет настроить редактор так, чтобы он выполнял такое форматирование автоматически.

Общий признак хорошего кода: его можно прочитать по телефону и он будет понятен на той стороне.

2. Требования к дизайну

Общий дизайн выполненной работы должен удовлетворять следующим требованиям:

1. Корректно обрабатывать ошибки при взаимодействии с внешним миром: ошибки ввода-вывода, некорректный пользовательский ввод и прочее. Полные перечень ошибок указан для каждой работы в условии.
2. Программы не должны содержать лишних сущностей: закомментированных участков кода, неиспользуемых переменных и функций, ...

3. Должно присутствовать разумное деление по файлам и правильно сформированные заголовочные файлы с минимальными зависимостями.

3. Общие требования к работам

Работы выполняются на языках программирования C/C++ (в зависимости от задания). Это означает, что код должен компилироваться и работать на любой платформе: Microsoft Windows, Linux, Mac OS X. Это автоматически означает, что все платформенно-зависимые вещи должны быть исключены.

Программы должны представлять собой консольные приложения, обрабатывающие параметры командной строки и использующие стандартные каналы ввода-вывода. Для каждой работы будет указано, что именно необходимо использовать. Тестирование заданий не предусматривает интерактивного взаимодействия. Ошибки ввода должны немедленно обрабатываться и программа должна завершиться с ненулевым кодом возврата, так как исправлять ввод при неинтерактивном взаимодействии невозможно. Стандартный набор кодов возврата:

0 Успешное завершение программы.

1 Некорректные входные данные или не удалось открыть указанный файл. Обычно, в стандартный поток ошибок должно быть выведено описание причины ошибки.

2 Внутренняя ошибка программы, например, не удалось выделить память. В стандартный поток ошибок также следует вывести описание ошибки.