РПОСУ. ЛР № 1. Основы языка C++

Дополнительные материалы (в ЛР объясняется необходимый минимум):

- Основы работы с командной строкой
- Перенаправление стандартных потоков ввода и вывода
- Документация к программе FC

Цель работы

1. Владеть базовыми конструкциями и типами языка C++.
2. Уметь работать в среде программирования CodeBlocks.
3. Уметь автоматически проверять программы по эталонному вводу и выводу.

Задание

1. Написать программу для построения гистограммы массива чисел.
2. Доработать программу в соответствии с вариантом.

Гистограмма — это наглядное графическое представление того, какие значения встречаются чаще или реже в исходных данных (как они распределены). Диапазон исходных данных делится на равные интервалы, для каждого интервала строится столбец. Высоты столбцов пропорциональны количеству значений, попавших в интервал. Таким образом сразу видно, какие значения встречаются чаще в целом (с точностью до ширины интервала) и насколько чаще по сравнению с другими (легко сравнить высоты визуально).

Гистограмма строится так: диапазон значений на входе делится на несколько равных интервалов (корзин, bins), подсчитывается количество чисел, попавших в каждый интервал, и для каждой корзины рисуется столбец, размер которого пропорционален количеству попавших в корзину чисел.

Например, на вход поступают оценки 10 студентов:

```
4 4 3 5 3 4 5 5 4 4
```

Пусть требуется построить гистограмму на три столбца. Диапазон значений на входе — от 3 до 5. Каждый из трех интервалов будет шириной

(5 − 3) / 3 = 0,67, то есть интервалы будут *[3; 3.67], [3.67; 4.34], [4.34; 5]*. В первую корзину попадут тройки (2 шт.), во вторую — четверки (5 шт.), в третью — пятерки (3 шт.). Результат:

```
2|**
5|*****
3|***
```

В данном случае в каждый столбец попадает только одно значение (3, 4 или 5), но в принципе, в один столбец могут попадать разные значения. Например, если для тех же оценок использовать два интервала, 4 и 5 попадут в один интервал:

```
2|**
8|********
```

Требования к выводу:

- Подписи к столбцам выровнены до трех знакомест (можно считать, что в корзину больше 999 чисел не попадет).
- Ширина всей гистограммы (подписи и звездочек в каждом столбце) должна укладываться в 80 символов. Если в корзину попало больше чисел, все столбцы нужно пропорционально сжать, чтобы выполнить условие.

Указания к выполнению работы

Алгоритм работы программы логично разделить на три этапа:

1. Ввод данных (считывание массива чисел и количества столбцов).
2. Обработка данных (расчет количества чисел, попавших в каждую корзину).
3. Вывод данных (отображение рассчитанных значений в виде гистограммы).

Ввод данных составляет:

1. Ввод количества значений (целого числа).
2. Ввод значений (цикл, заполняющий массив).
3. Ввод количества корзин (целого числа).

Обработка заключается в том, чтобы массив чисел преобразовать в массив их количеств в каждой корзине. Для этого нужно рассмотреть каждое число, узнать номер корзины, в которую оно попадает, и увеличить счетчик чисел для этой корзины.

Вывод данных имеет смысл выполнить в несколько подходов: сначала элементарно, чтобы убедиться в правильности обработки, затем реализовать требования к отображению.

1. Отображать для каждой корзины количество чисел в ней, ось и такое же количество звездочек, сколько чисел в корзине.
2. Выровнять подписи до трех знакомест.
3. Масштабировать высоты столбцов, чтобы уложиться в 80 символов.

Ввод данных

Входные данные (переменные): количество чисел, числа, количество корзин.

В C++ есть специальный тип `size_t`, подходящий для размеров массивов: целое неотрицательное число с широким диапазоном значений. Целесообразно использовать его для количества чисел:

```
size_t number_count;
```

Имя переменной сообщает, что в ней хранится: number count по-английски «количество чисел». Всем переменным нужно давать осмысленные имена (исключения: счетчики циклов `i`, `j`, … и математические формулы). Программы чаще читаются, чем пишутся — их понятность важнее размера. Скорости набора текста помогает редактор, подсказывающий имена по мере ввода.

Ввод стандартный:

```
cout << "Enter number count: ";
cin >> number_count;
```

Отображение кириллицы в консоли Windows требует ухищрений. Применять их не нужно, во всех ЛР достаточно англоязычного вывода.

Массив значений состоит из действительных чисел, выберем для них тип `double`. Размер массива определяется во время работы программы переменной `number_count`, то есть это динамический массив. Его можно реализовать через `new[]`/`delete[]`, но в современном C++ принято использовать вектор, в данном случае — `vector<double>`.

Для использования `vector<T>` нужно подключить часть стандартной библиотеки:

```
#include <vector>
```

При объявлении переменной `numbers` (числа) типа `vector<T>` можно сразу указать размер:

```
vector<double> numbers(number_count);
```

Очевидно, это нужно делать после ввода `number_count`. Если бы `numbers` была объявлена до этого, ей не нужно было бы передавать аргумент в скобках; вместо этого после ввода `number_count` нужно было бы изменить размер:

```
numbers.resize(number_count);
```

Однако безопаснее объявлять переменные как можно ближе к месту первого использования: меньше риск случайно обратиться к ним до инициализации. Например, если объявить `numbers` в начале программы, ничто не помешает обратиться к его элементам до вызова `resize()`, что приведет к ошибке.

Числа вводятся в `numbers` стандартным циклом `for` со счетчиком. Код нужно написать самостоятельно.

Количество корзин `bin_count` целесообразно сделать того же типа, что и `number_count` и вводить так же.

Обработка данных

Необходимо для каждой корзины подсчитать количество попавших в нее чисел, то есть заполнить массив счетчиков. Тип счетчика — `size_t`, потому что это по сути такое же количество, как количество чисел. Их массив имеет размер `bin_count`, а начальные значения в нем — нули:

```
vector<size_t> bins(bin_count);
```

Как по значению элемента определить номер корзины, куда он попадает?

Определение индекса корзины по значению элемента

Нужно подсчитать, сколько чисел попало в каждую корзину. Для этого можно определить номер корзины, в которую попадает каждое число, и увеличить счетчик этой корзины.

Каждая корзина представляет числа в одном из интервалов, на которые равномерно разбит диапазон исходных чисел. Например, для чисел `4 5 3 4 5 4 3 4 5 4` и трех корзин:

```
         4
         4
         4     5
   3     4     5
   3     4     5
|-----|-----|-----|
3.00  3.67  4.34  5.00
```

Нижняя граница будет равна минимальному из чисел (обозначим его `min`), верхняя — максимальному `max`. Каждая следующая граница отстоит от предыдущей на размер корзины `bin_size`:

```
double bin_size = (max - min) / bin_count;
```

В общем случае:

```
|------------|------------|------------|------------|
min          min+1*step   min+2*step   min+3*step   max
:            :
:<---------->:
   bin_size
```

Пусть `min` и `max` найдены, `bin_size` рассчитан по формуле выше. Остается для каждого `i`-го числа проверить каждую `j`-ю корзину. Если число попадает между границ этой корзины, то счетчик попавших в корзину чисел увеличивается. Прочие корзины можно уже не просматривать.

Пример 1. При *i = 0* рассматривается число 4 из списка выше. При *j = 0* рассматривается интервал *[3.00; 3.67),* в который 4 не входит. При *j = 1* рассматривается интервал *[3.67; 4.34),* куда 4 попадает. Следовательно, счетчик `bins[1]` нужно увеличить, а цикл по `j` можно прекратить.

Пример 2. При *i = 1* рассматривается число 5. Оно не входит ни в один из интервалов, даже *[4.34; 5.00),* потому что правая граница не учитывается. Этот особый случай нужно опознать и увеличить счетчик последней корзины.

```
for (size_t i = 0; i < number_count; i++) {
    bool found = false;
    for (size_t j = 0; (j < bin_count - 1) && !found; j++) {
        auto lo = min + j * bin_size;
        auto hi = min + (j + 1) * bin_size;
        if ((lo <= numbers[i]) && (numbers[i] < hi)) {
            bins[j]++;
            found = true;
        }
    }
    // цикл по numbers не закончился!
```

Особый случай `number == max` из примера 2 можно опознать по `found == false` после цикла:

```
    if (!found) {
        bins[bin_count - 1]++;
    }
} // конец цикла по numbers
```

Отметим оформление кода:

- вокруг операторов (`=`, `<`, `>`), после ключевых слов (`for`, `if`) и перед фигурными скобками (`{`) стоят пробелы;
- блоки кода (тело цикла, инструкции под условиями) выделены отступами.

Нижняя и верхняя границы интервала обозначены `lo` (low) и `hi` (high), это типовые имена переменных, как `i`, `j` для счетчиков.

Как и имена переменных, форматирование кода (code style) помогает понимать смысл программы (ее структуру) и страхует от ошибок. Чтобы автоматически отформатировать код в CodeBlocks,

нужно щелкнуть по тексту правой кнопкой мыши и выбрать *Format (use AStyle)* (если выделить фрагмент, отформатирован будет только он).

Определение диапазона чисел в массиве

Для расчета индексов столбцов нужно найти наибольший и наименьший элементы в массиве. Разберем эту простую задачу подробно, чтобы изучить диапазонный цикл `for` (range-based for loop). Формулировка решения: *для каждого* массива чисел сравнить его с максимумом и минимумом, при необходимости скорректировать максимум или минимум. Заметим, что решение не оперирует индексами в массиве — только значением очередного элемента. Для выражение этой идеи в C++ есть диапазонный цикл `for`:

```
double min = numbers[0];
double max = numbers[0];
for (double x : numbers) {
    if (x < min) {
        min = x;
    }
    else if (x > max) {
        max = x;
    }
}
```

Запись `for (double x : numbers) { ... }` означает: выполнить тело цикла *для каждого* элемента `x` (типа `double`) из массива `numbers`, то есть в точности соответствует логике решения.

Вывод данных

Этап 1: минимальный работающий вариант

Для каждого элемента `bin` массива `bins` нужно вывести значение `bin`, символ `|` и *bin* звездочек (внутренним циклом со счетчиком); после звездочек нужен перевод строки. Код нужно написать самостоятельно.

На этом этапе можно проверить работу программы: ввести данные из примера (десять чисел, три корзины) и визуально сопоставить гистограммы.

На последующих этапах нужно будет проверять работу программы на десятках чисел, вводить которые долго и чревато ошибками. Можно записать числа в файл и вставлять в консоль, но сравнивать результаты (считать звездочки) не легче. Проблема усугубляется, если нужно проводить много тестов.

Автоматическая проверка по эталонному вводу и выводу

Можно полностью автоматически вводить данные в программу, сохранять ее вывод и сравнивать с эталоном, получая простой ответ: пройден ли тест.

Эталонный ввод при этом читается из файла, вывод записывается в файл, который затем сравнивается с файлом эталонного вывода. При этом не требуется добавлять в программу работу с файлами и логику проверки, если знать, как устроен ввод и вывод, и уметь пользоваться стандартными утилитами.

Дальнейшая работа ведется в консоли из каталога с файлом `*.exe`, в CodeBlocks это может быть `bin\Debug`. При затруднениях в работе с консолью можно воспользоваться руководством.

Командная строка Windows

Командная строка (терминал) запускается через *Win+R*, `cmd` или путем ввода `cmd` в строку адреса в «Проводнике» и нажатия *Enter*. Текст `C:\>` слева от курсора называется *приглашением (prompt)*. Приглашение показывает текущий каталог — корень диска C. Перейти в другой каталог можно командой `cd`, например, `cd lab01`. Если нужно перейти на другой диск, добавляется *ключ (опция)* `/d`, например: `cd /d L:\A-01-18\username`. Чтобы не вводить путь вручную, можно нажимать *Tab* после ввода первых символов имени каталога, и Windows дополнит путь. Если нужно повторить одну из предыдущих команд, стрелки вверх и вниз проматывают историю.

Стандартные потоки и их перенаправление

Обычно для простоты говорят, что ввод происходит с клавиатуры, а вывод — на экран. На самом деле ввод происходит из особого устройства — *стандартного ввода (standard input, stdin),* а вывод поступает на устройство *стандартного вывода (standard output, stdout)*. По умолчанию стандартный ввод связан с клавиатурой, а вывод — с терминалом (окном консоли в Windows). Однако можно при запуске программы указать, что стандартным вводом для нее будет не клавиатура, а файл *(перенаправление ввода, input redirection):*

```
C:\lab01> lab01.exe < 01-example.input.txt
Enter number count: Enter numbers: Enter bin count: 2|**
5|*****
3|***
```

Внимание. Здесь и далее примеры работы с командной строкой включают приглашение `C:\lab01>`, команду и ее вывод. Приглашение вводить не нужно, это просто стандартный формат записи. Таким образом, нужно ввести `lab01.exe < 01-example.input.txt`, в ответ ожидается текст на второй строке и далее.

Готовые файлы, которые нужно скачать или создать:

- эталонный ввод 01-example.input.txt
- эталонный вывод 01-example.expected.txt

Видно, что гистограмма строится правильно, но картину портят приглашения ввода (`Enter number count` и прочие).

Аналогично можно направить стандартный вывод в файл:

```
C:\lab01> lab01.exe < 01-example.input.txt > 01-example.actual.txt
```

Вывода на терминал нет — он весь в `01-example.actual.txt`, и если его просмотреть, окажется, что он соответствует предшествующему выводу.

Как избавиться от приглашений, которые не нужны в режиме автоматических тестов? Проблема заключается в том, что у программы есть значимый, информативный вывод (собственно гистограмма), а есть декоративный вывод (приглашения). Только самой программе «известно», где какой вывод — без ее модификации не обойтись.

Помимо стандартного вывода существует *стандартный вывод ошибок (stderr).* Такое название сложилось исторически, а на практике принято писать в него декоративный вывод. Этот поток доступен в C++ как `cerr`:

```
cerr << "Enter number count: ";
```

Нужно самостоятельно заменит вывод приглашений (но не гистограммы) в `cout` на их вывод в `cerr`.

Если теперь запустить программу как обычно (без перенаправления), ее работа внешне не изменится, потому что стандартный вывод ошибок по умолчанию тоже связан с терминалом. Если же перенаправить вывод в файл, записанное в `cerr` появится в терминале, но не в файле `01-example.actual.txt`:

```
C:\lab01> lab01.exe < 01-example.input.txt > 01-example.actual.txt
Enter number count: Enter numbers: Enter bin count:
```

Чтобы убрать декоративный вывод при автоматических тестах, можно направить его в специальное устройство `NUL`, которое поглощает любой вывод в него:

```
C:\lab01> lab01.exe < 01-example.input.txt > 01-example.actual.txt 2>NUL
```

Вывода в терминал при этом нет никакого, хотя программа успешно работает.

При желании с перенаправлением можно ознакомиться подробнее.

Сравнение файлов

Программа `fc` (file compare) позволяет построчно сравнить файл вывода программы `01-example.actual.txt` с файлом `01-example.expected.txt`, содержащим эталонный вывод:

```
C:\lab01> fc 01-example.actual.txt 01-example.expected.txt
Сравнение файлов 01-example.actual.txt и 01-EXAMPLE.EXPECTED.TXT
FC: различия не найдены
```

Если бы были отличия, `fc` могла бы показать отличающиеся строки, а с ключом `/N` также и их номера (справка):

```
C:\lab01> fc /N 01-example.actual.txt 02-alignment.expected.txt
Сравнение файлов 01-example.actual.txt и 02-ALIGNMENT.EXPECTED.TXT
***** 01-example.actual.txt
    1:  2|**
    2:  5|*****
    3:  3|***
***** 02-ALIGNMENT.EXPECTED.TXT
    1:    2|**
    2:    5|*****
    3:    3|***
*****
```

BAT-файлы

Чтобы не вводить каждый раз команды вручную, их можно записать в файл с расширением `*.bat` и запускать как программу из командной строки:

```
lab01.exe < 01-example.input.txt > 01-example.actual.txt 2>NUL
fc /N 01-example.actual.txt 01-example.expected.txt
```

Чтобы запускать файл из «Проводника» и при ошибках окно не закрывалось, можно к последней строки добавить `|| pause`.

Этап 2. Выравнивание подписей столбцов

Требуется количество чисел, попавших в каждый столбец, дополнять при выводе пробелами так, чтобы суммарно подпись занимала четыре знакоместа (символа). Хотя в C++ и есть средства форматирования, реализуем выравнивание вручную:

1. Если число меньше 100, вывести пробел (он займет место разряда сотен).
2. Если число меньше 10, вывести пробел (он займет место разряда десятков).
3. Вывести число.

Необходимо написать код самостоятельно и автоматически проверить его:

- эталонный ввод 02-alignment.input.txt;
- эталонный вывод 02-alignment.expected.txt.

Этап 3. Масштабирование столбцов гистограммы

Поскольку ограничена ширина всей гистограммы (включая 3 цифры подписи и ось шириной 1 символ), ограничение на длину столбца будет $80 - 3 - 1 = 76$ символов.

Если в корзине самым большим количеством чисел не больше 76, масштабирование не нужно.

Для масштабирования из количества чисел `count` в каждой корзине нужно получить ее высоту `height` (количество звездочек). Масштабирование должно работать так, чтобы самый высокий столбец имел 76 звездочек, следовательно, для него `height = 76 * 1.0`. Для прочих корзин вместо 1,0 должен быть множитель-доля количества чисел в этой корзине от максимального количества: `height = 76 * count / max_count`. Однако напрямую эту формулу записать на C++ нельзя: деление целых чисел `count` и `max_count` даст целое же число.

Необходимо указать компилятору, что `count` нужно рассматривать как дробное. Это называется приведением типов (type cast):

```
size_t height = 76 * (static_cast<double>(count) / max_count);
```

Выражение `static_cast<T>(x)` означает: рассматривать выражение `x` как имеющее тип `T`.
Можно встретить другую форму записи, так называемое приведение в стиле C (C-style cast): `((double)count)`. Почему в C++ более громоздкий синтаксис? Дело в том, что приведение типов — это место в программе, где программист берет на себя ответственность, что преобразование имеет смысл, поэтому лучше, когда оно резко выделяется в тексте программы.

Заметим, что в данном примере можно было бы обойтись вообще без приведения типов, если сделать `max_count` типа `double`, однако приведение типов часто встречается на практике — необходимо уметь его делать.

Константа 76 используется как минимум в двух местах, что плохо. во-первых, если потребуется поменять ее, придется искать и редактировать все эти места. Во-вторых, при чтении кода будет непонятен ее смысл. По последней причине такие числа в коде называются магическими константами. Их нужно выносить в неизменяемые переменные с понятными именами или комментариями. Обычно их размещают в самом начале программы:

```
const size_t SCREEN_WIDTH = 80;
const size_t MAX_ASTERISK = SCREEN_WIDTH - 3 - 1;
```

Итак, требуется самостоятельно реализовать:

1. Поиск наибольшего количества чисел в одной корзине (можно совместить в подсчетом чисел в корзинах).
2. Масштабирование столбцов.

Результат необходимо автоматически проверить:

- эталонный ввод 03-scaling.input.txt
- эталонный вывод 03-scaling.expected.txt

Варианты индивидуальных заданий

Решение должно включать: код программы, файлы эталонного ввода и вывода, BAT-файл для автоматической проверки.

Вариант 1

Дайте пользователю возможность задавать произвольную ширину гистограммы вместо 80 символов. Ширину менее 7, более 80 или менее трети количества чисел считайте некорректной — предлагайте пользователю ввести ее заново в этом случае с указанием причины.

Вариант 2

Если пользователь вводит 0 как число столбцов, рассчитывайте число столбцов автоматически по эмпирической формуле $K = \sqrt{N}$, а если получилось $K > 25$, пересчитайте по правилу Стёрджеса: для N чисел количество столбцов $K = 1 + \lfloor log_2 N \rfloor$. Печатайте, по какой формуле был сделан выбор и сколько столбцов выбрано.

Указание. См. функции `sqrt()` и `log2` из `<cmath>`.

Вариант 3

Дайте пользователю возможность задавать высоту гистограммы H строк. Если количество столбцов K в $C = \lfloor H/K \rfloor$ раз меньше H, столбцы должны занимать по C строк.

Пример. Выбрано $H = 6, K = 3 \Rightarrow C = 2$, гистограмма:

```
 8|********
  |********
11|***********
  |***********
 6|******
  |******
```

Вариант 4

Вместо количества элементов сделайте подписью столбца процент элементов, попавших в столбец, как целое двузначное число с `%` в конце.

Вариант 5

Отображайте гистограмму зеркально, например:

```
   ********|  8
***********| 11
     ******|  6
```

Вариант 6

Дайте пользователю возможность выбора символов для столбцов «рисунка», линии оси (`|` в примерах) и для выравнивания подписей. Например, при выборе соответственно `|`, пробела и `0`:

```
008 ||||||||
011 ||||||||||
006 ||||||
```

Не позволяйте вводить символы табуляции и перевода строк, печатайте любое сообщение со словом «ERROR» и завершайте программу при этом.

Вариант 7

Вычислите среднюю высоту столбца. Если столбец ниже, доведите его высоту до средней символами `-`. Если столбец выше, выводите часть, превышающую среднюю высоту, символами `+`. Пример (средняя высота — 8 звездочек):

```
 8|********
11|********+++
 6|******--
```

Вариант 8

После подсчета количеств значений в столбцах, замените их нарастающим итогом, начиная с первого столбца. При отображении соблюдайте те же правила, что и ранее. Пример для исходного графика с высотами 1-3-7-11-6-4-1:

```
 1|*
 4|****
11|***********
22|**********************
28|****************************
32|********************************
33|*********************************
```

Вариант 9

В каждом столбце, если предыдущий столбец ниже, вместо `*` используйте `^` на высоте предыдущего столбца. Аналогично для следующего столбца, но `v`. Если соседние столбцы оба ниже текущего и равны, используйте `N`. Пример:

```
 1|*
 3|^**
 7|**^*****
11|*****v^****
 6|***v**
 4|v***
 1|*
```

Вариант 10

Отображайте гистограмму вертикально без подписей, например:

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

Указание. Можно воспользоваться следующей логикой: проходить по всем столбцам и печатать `*`, если высота столбца больше номера строки, или пробел, если нет — и так до тех пор, пока на очередной строке печатается хотя бы одна звездочка.

Вариант 11

Добавьте рамку вокруг гистограммы. Добавьте учет линий рамки, чтобы общая ширина «изображения» не превышала 80 символов. Иллюстрация результата:

```
+----------------+
|  8|********    |
| 11|********** |
|  6|******      |
+----------------+
```

Вариант 12

Добавьте на ось подписей границы столбцов. Например, если в первый столбец отнесены элементы от наименьшего до 1,23, во второй — от 1,23 до 2,34 и т. д., желаемый результат:

```
     8|********
1.23
    11|***********
2.34
     6|******
```

Ширину места для подписей столбцов нужно увеличить, как на иллюстрации.

Вариант 13

После вывода гистограммы запрашивайте у пользователя, доволен ли он результатом. Если ответ отрицательный, позвольте ввести новое количество столбцов и перестройте гистограмму. Процесс может повторяться сколько угодно раз.

Вариант 14

Сделайте подписи к столбцам текстовыми. После ввода количества столбцов *K* пользователь должен ввести *K* строк (возможно, с пробелами), которые будут подписями к соответствующим столбцам. При выводе гистограммы вместо высоты каждого столбца нужно печатать его подпись. Подписи должны быть выровнены по правому краю на ширину самой длинной из них.

Вариант 15

Добавьте горизонтальную шкалу под гистограммой. Шкалу нужно разбить на интервалы, размер которых от вводит пользователь. Допустимы размеры от 4 до 9, при некорректном вводе печатайте сообщение со словом «ERROR» и завершайте работу программы. Под нулевой, первой и последней отметкой шкалы требуется напечатать соответствующие числа. Шкала должна быть во всю ширину гистограммы. Пример для интервала размером 6:

```
 8|********
14|**************
12|************
  |-----|-----|-----|
  0     6           18
```

Вариант 16

Перед построением гистограммы удалите из входного массива все повторяющиеся (не обязательно подряд) элементы и напечатайте результат.

Указание. Удалить `xs[i]` можно так: `xs.erase(xs.begin() + i)`.

Вариант 17

После ввода количества чисел предлагайте пользователю генерировать их. При положительном ответе заполните исходный массив при помощи функции `rand()`: каждый элемент должен быть суммой 12 ее результатов.

Указание. В начале программы добавьте `srand(time(0))`, чтобы случайные числа отличались между запусками программы (аналог `Randomize()` в Pascal). Для составления эталонного вывода замените `time(0)` на 42.

Вариант 18

Избавьте программу от предположения о наибольшем возможном количестве чисел в столбце. Находите наибольшее и используйте это значение, чтобы выровнять подписи по правому краю, не расходуя при этом лишних знакомест.

Козлюк Д. А., Мохов А. С., Василькова П. Д. для кафедры Управления и информатики НИУ «МЭИ», 2021 г.