Задачи:

Часть A – на оценку хор

1. Основные команды командной строки linux: cd, ls, pwd, cp, mv, mkdir, текстовый редактор nano, компилятор gcc.
   Что нужно: Уметь создавать файлы и папки, компилировать исходный код.
   Пример задачи: Создать файл исходного кода программы, которая будет считывать числа из входного файла, складывать их и записывать результат в другой файл. Создать файл входных данных. Скомпилировать программу. Всё это, пользуясь только командами командной строки. Для тех, у кого нет возможности использовать linux:
   www.tutorialspoint.com/unix_terminal_online.php

2. Управляющие конструкции: if, else, while, for, break, continue. Преобразование типов. Работа с массивами.
   Что нужно: Уметь использовать все перечисленные управляющие конструкциями, преобразовывать типы, работать с массивами. Уметь решать простые алгоритмические задачи.
   Пример задачи: Напечатать все простые числа от n до m, n и m даны.

3. Функции и указатели
   Что нужно: Уметь писать прототипы функции, писать описание функций, правильно вызывать функции. Знать что такое указатель, как его объявлять и использовать. Передача по ссылке и передача по значению.
   Пример задачи: Написать и использовать функцию нахождения факториала, рекурсивную и итеративную(т.е. используя цикл).

4. Структуры
   Что нужно: Уметь объявлять и использовать структуры и указатели на структуры.
   Пример задачи: Описать структуру профиля социальной сети. Структура должна содержать поля: id пользователя(целое число), имя, фамилия, возраст, дата создания профиля и список друзей(массив из разных id). Создать массив таких структур и инициализировать все структуры в массиве. Написать функции для работы с такими структурами:

   (a) функция, которая добавляет 1 профиль в массив

   (b) функция, которая изменяет возраст, заданный в профиле пользователя

   (c) функция, которая делает 2-х пользователей друзьями

   (d) функция, которая по данному id пользователя, выводит имена и фамилии всех его друзей.

5. Простые алгоритмы и структуры данных. O(n) нотация.
   Что нужно: Уметь писать простейшие алгоритмы сортировки(пузырьком, вставками, выбором), алгоритм бинарного поиска. Уметь определять сложность алгоритма. Знать сложности всех пройденных алгоритмов сортировки (пузырьком, вставками, выбором, быстрая, сортировка слиянием, цифровая). Знать такие структуры данных, как стек, очередь, список. Знать или уметь выводить сложности операций с проиденными структурами данных: вставка в массив, стек, список; удаление из массива, стека, списка; поиск по массиву и списку.
   Пример задачи: Написать сортировку пузырьком.

Часть B – на оценку отл

1. Динамическое выделение памяти. malloc и free.
   Что нужно: Знать что такое стек(stack) и куча(heap) процесса, их особенности и различия. Уметь динамически выделять память с помощью malloc и удалять с помощью free. Уметь пользоваться valgrind.
   Пример задачи: Описать структуру профиля социальной сети. Структура должна содержать поля: id пользователя(целое число), имя, фамилия, возраст, дата создания профиля и список друзей(массив из разных id). Динамически выделить память под массив из таких структур. В каждой структуре динамически выделить память под массив друзей. Написать функции для работы с такими структурами:

   (a) функция, которая добавляет 1 профиль в массив. Если не хватает места в массиве, то нужно выделить больше места.

   (b) функция, которая делает 2-х пользователей друзьями. Если не хватает места в массиве друзей, то нужно выделить больше места.

   (c) функция, которая по данному id пользователя, выводит имена и фамилии всех его друзей.

2. Строки
   Что нужно: Уметь объявлять строки. Знать как строки реализованы в языке C. Уметь использовать основные стандартные функции для работы со строками: strlen, strcpy, strcat, strcmp, strchr, strtol.
   Пример задачи: Найти сумму всех чисел в строке. Например ответ для строки "1hwe6j32yz4"должен быть равен 43.

3. Файлы и аргументы командной строки
   Что нужно: Открывать файлы. Использовать функции fprintf, fscanf. Посимвольное чтение/запись файла – функции fputc, fgetc. Бинарное чтение/запись – fread, fwrite. Уметь использовать в программе аргументы командной строки
   Пример задачи: Написать свой аналог программы wc. Программа должна принимать на вход имя файла как аргумент командной строки. Нужно вывести количество строк, слов и символов в этом файле. Словом считается любая последовательность символов, разделённая пробелами, переносами строк('\n') или знаками табуляции ('\t').

4. Алгоритмы сортировки
   Что нужно: Уметь писать алгоритмы сортировки(пузырьком, вставками, выбором, **быстрая, сортировка слиянием**). Знать сложности всех пройденных алгоритмов сортировки (пузырьком, вставками, выбором, быстрая, сортировка слиянием, цифровая). Уметь использовать стандартный функцию сортировки qsort.
   Пример задачи: Написать свой алгоритм быстрой сортировки.

5. Структура данных список
   Что нужно: Знать что такое список, как он реализуется в языке C. Как реализуются функции нахождения длины списка, вставки элемента в начало/конец, удаления элемента из начала/конца, поиска в списке.
   Пример задачи: Обратить список так, чтобы первый элемент списка стал последним, а последний первым.

Часть C – на оценку отл(9-10)

Только для тех, у кого $>$ 70% баллов.
Можно делать вместе. Вспомагательный код и входные данные расположены здесь:
github.com/v-biryukov/cs_mipt_faki/tree/master/term1/final_tasks

AABB – деревья

Задача заключается в нахождении пересечения 2-х ломанных. Ломаные представляют собой совокупность соединённых друг за другом отрезков. Простейший способ решения этой задачи: полный перебор. Для этого нужно просто проверить на пересечение каждый отрезок одной ломанной с каждым отрезком другой. Про то, как найти пересечение 2-х отрезков можно посмотреть, например, здесь:
stackoverflow.com/questions/563198/how-do-you-detect-where-two-line-segments-intersect
Однако полный перебор может быть очень затратным если число отрезков в ломаной велико. Гораздо боле быстрый способ нахождения пересечений таких ломаных это использование так называемых AABB-деревьев. AABB расшифровывается как axis-aligned bounding box, то есть параллельный осям ограничивающий прямоугольник. Такой прямоугольник очень удобен, так как можно очень просто найти пересечение 2-х таких прямоугольников. К тому же они занимают очень мало места: всего 2-х точек достаточно, чтобы задать AABB. AABB-дерево же представяет собой обычное алгоритмическое дерево, только в каждом узле такого дерева будут храниться 2 точки, задающие AABB. В нашем случае мы ещё будем хранить динамический массив из всех сегментов(отрезков), которые находятся внутри этого ограничивающего прямоугольника.

```
// Описание точки
typedef struct _point
{
    double x, y;
} Point;

// Описание сегмента
typedef struct _segment
{
    int p1, p2; // номера точек, которые будут храниться в общем массиве
} Segment;
```

```
// Описание узла AABB дерева
typedef struct _AABB_node
{
    // lb -- нижняя левая точка AABB  -- left bottom
    // rt -- верхняя правая точка AABB  --  right top
    Point lb, rt;
    // segments -- массив из сегментов, принадлежащих этому узлу
    int number_of_segments;
    Segment * segments;

    // указатели на детей
    struct AABB_node_ * ch0;
    struct AABB_node_ * ch1;
} AABB_node;
```

Алгоритм нахождения пересечений с помощью AABB-деревьев заключается в следующем:

1. Сначала строим дерево для одной из ломаных.
   (a) Корень этого дерева будет содержать ограничивающий прямоугольник который охватывает все отрезки этой ломаной. Нужно найти и задать координаты такого прямоугольника.
   (b) Затем нужно определиться по какой из координат мы будем делить ломаную. Если ограничивающий прямоугольник длиннее по оси Ox, то будем делить по оси Ox. Если длиннее по оси Oy, то по Oy.
   (c) Создаём детей корня дерева так, чтобы каждый из них ограничивал примерно половину сегментов.
   (d) Рекурсивно повторяем то же самое для детей пока узлы дерева не будут ограничивать небольшое число сегментов
2. Строим аналогичное дерево для второй ломаной.
3. Ищем пересечение двух ломаных.
   (a) Сначала проверяем пересекаются ли ограничивающие прямоугольники корней 2-х деревьев.
   (b) Если они пересекаются, то рекурсивно попарно проверяем пересекаются ли их дети.
   (c) Когда дойдём до узлов, не имеющих детей, то проверяем пересекаются ли кусочки ломанных ограничивающиеся этим узлом с помощью перебора. Так как эти узлы будут хранить малое количество сегментов, то перебор не займёт много времени.

Код частичного решения задачи:
github.com/v-biryukov/cs_mipt_faki/blob/master/term1/final_tasks/aabb_tree/example/AABB_tree_2d.c

Входные данные:
github.com/v-biryukov/cs_mipt_faki/blob/master/term1/final_tasks/aabb_tree/curves

Скачать всё сразу можно зайдя на:
github.com/v-biryukov/cs_mipt_faki/

и нажав на "Clone or download> "Download ZIP".

Рис. 1: На рисунки изображены 2 ломаные, состоящие из большого числа отрезков. Первая ломаная представляет собой контур Африки, вторая – контур дракона. Видно, что эти 2 ломаные не пересекаются.