Задания к работе 2 по прикладному программированию.

Все задания реализуются на языке программирования *C* (стандарт *C99* и выше).

Реализованные в заданиях приложения не должны завершаться аварийно.

Во всех заданиях запрещено использование глобальных переменных (включая *errno*).

Во всех заданиях запрещено использование оператора безусловного перехода (*goto*).

Во всех заданиях запрещено пользоваться функциями, позволяющими завершить выполнение приложения из произвольной точки выполнения, вне контекста исполнения функции *main*.

Во всех заданиях при реализации необходимо разделять контексты работы с данными (поиск, сортировка, добавление/удаление, модификация и т. п.) и отправка данных в поток вывода / выгрузка данных из потока ввода.

Во всех заданиях все параметры функций и вводимые (с консоли, файла, командной строки) пользователем данные должны подвергаться валидации в соответствии с типом валидируемых данных, если не сказано обратное; валидация должна зависеть от типа данных и логики применения этих данных для выполнения целевой подзадачи. При передаче аргументов приложению в командную строку, их количество также должно валидироваться.

Во всех заданиях необходимо контролировать ситуации с невозможностью [пере]выделения памяти; во всех заданиях необходимо корректно освобождать всю выделенную динамическую память.

Все ошибки, связанные с операциями открытия файла, должны быть обработаны; все открытые файлы должны быть закрыты.

Во всех заданиях запрещено использование глобальных переменных.

Во всех заданиях при реализации функций необходимо обеспечить возможность обработки ошибок различных типов на уровне вызывающего кода при помощи возврата целевых результатов функции через параметры функции и возврата из функции значения (либо типа *int*, либо перечислимого типа (*enum*)), репрезентирующего статус-код функции, в целях обработки последнего в вызывающем коде через оператор *switch/case* либо через управляющие конструкции языка *if / else if / else*. Возвращаемые статус-коды функций необходимо продумать самостоятельно так, чтобы были покрыты всевозможные ошибки времени выполнения функций.

Во всех заданиях сравнение (на предмет эквивалентности или отношения порядка) вещественных чисел на уровне функции должно использовать значение эпсилон, которое является параметром этой функции.

Во всех заданиях при реализации функций необходимо максимально ограничивать возможность модификации (если она не подразумевается) передаваемых в функцию параметров (используйте ключевое слово *const*).

1. Реализовать функцию перевода числа из десятичной системы счисления в систему счисления с основанием 2^r, $r = 1, ..., 5$. При реализации функции разрешается использовать битовые операции и операции обращения к памяти, запрещается использовать стандартные арифметические операции. Продемонстрируйте работу реализованной функции.

2. Через аргументы командной строки программе подаются флаг (первым аргументом), строка (вторым аргументом); и (только для флага -c) целое число типа *unsigned int* и далее произвольное количество строк. Флаг определяет действие, которое необходимо выполнить над переданными строками:
   - -l подсчёт длины переданной строки, переданной вторым аргументом;
   - -r получить новую строку, являющуюся перевёрнутой (reversed) переданной вторым аргументом строкой;
   - -u получить новую строку, идентичную переданной вторым аргументом, при этом каждый символ, стоящий на нечётной позиции (первый символ строки находится на позиции 0), должен быть преобразован в верхний регистр;
   - -n получить из символов переданной вторым аргументом строки новую строку так, чтобы в начале строки находились символы цифр в исходном порядке, затем символы букв в исходном порядке, а в самом конце – все остальные символы, также в исходном порядке;
   - -c получить новую строку, являющуюся конкатенацией второй, четвёртой, пятой и т. д. переданных в командную строку строк; строки конкатенируются в псевдослучайном порядке; для засеивания генератора псевдослучайных чисел функцией *srand* используйте *seed* равный числу, переданному в командную строку третьим аргументом.

   Для каждого флага необходимо реализовать соответствующую ему собственную функцию, выполняющую действие. Созданные функциями строки должны располагаться в выделенной из динамической кучи памяти. При реализации функций запрещено использование функций из заголовочного файла *string.h*. Продемонстрируйте работу реализованных функций.

3. Реализовать функцию с переменным числом аргументов, выполняющую возврат ресурсов в контекст запроса:
   - возврат в кучу ранее выделенной динамической памяти
   - закрывающую ранее открытые под файлы файловые переменные

   Для конфигурирования действий по возврату ресурса, для каждого ресурса в функцию необходимо передать два параметра: указатель на ресурс и флаг действия. При получении из списка аргументов переменной длины указателя на ресурс со значением NULL, обработка списка аргументов переменной длины должна быть прекращена. Передача количества аргументов в функцию при этом не допускается. Продемонстрируйте работу реализованной функции.

4. Реализовать функцию с переменным числом аргументов, принимающую в качестве входных параметров подстроку и пути к файлам. Необходимо чтобы для каждого файла функция производила поиск всех вхождений переданной подстроки в содержимом этого файла. В подстроку могут входить любые символы (обратите внимание, что в подстроку также могут входить символы пробела, табуляций, переноса строки, в неограниченном количестве; пустая подстрока также допустима). При реализации запрещается использование функций из заголовочного файла *string.h*. Продемонстрируйте работу функции, также организуйте наглядный вывод результатов работы функции: для каждого файла, путь к которому передан как параметр Вашей функции, для каждого вхождения подстроки в содержимое файла необходимо вывести номер строки (индексируется с 1) и номер символа в строке файла (индексируется с 1), начиная с которого найдено вхождение подстроки.

5. Реализовать функцию с переменным числом аргументов, принимающую координаты (вещественного типа, пространство двумерное) вершин многоугольника и определяющую, является ли этот многоугольник выпуклым. Продемонстрируйте работу функции.

6. Реализовать функцию с переменным числом аргументов, находящую значение многочлена степени *n* в заданной точке. Входными параметрами являются точка (вещественного типа), в которой определяется значение многочлена, степень многочлена (целочисленного типа), и его коэффициенты (вещественного типа, от старшей степени до свободного коэффициента в порядке передачи параметров функции слева направо). Продемонстрируйте работу функции.

7. Реализовать функцию с переменным числом аргументов, находящую среди переданных строковых представлений целых неотрицательных чисел, заданных в системе счисления с основанием *base*, передаваемым как параметр функции, чисел, являющихся в системе счисления с основанием *base* числами Капрекара. Продемонстрируйте работу функции.

8. Реализовать функцию с переменным числом аргументов, вычисляющую сумму переданных целых неотрицательных чисел в заданной системе счисления с основанием *[2..36]*. Параметрами функции являются основание системы счисления, в которой производится суммирование, количество переданных чисел, строковые представления чисел в заданной системе счисления. Десятичное представление переданных чисел может быть слишком велико и не поместиться во встроенные типы данных; для решения возникшей проблемы также реализуйте функцию «сложения в столбик» двух чисел в заданной системе счисления, для её использования при реализации основной функции. Результирующее число не должно содержать ведущих нулей. Продемонстрируйте работу функции.

9. Реализовать функции *overprintf*, *overfprintf* и *oversprintf*, поведение которых схоже с поведением стандартных функций *printf*, *fprintf* и *sprintf*, то есть эти функции имеют одинаковый прототип и логику работы, но в ваших функциях помимо стандартных флагов для функций семейства *printf* (%d, %u, %f, %lf, %c, %s, %o, %x, %X) определены следующим образом дополнительные флаги:
    - %Ro – печать в поток вывода целого числа типа *int*, записанного римскими цифрами;
    - %Zr – печать в поток вывода цекендорфова представления целого числа типа *unsigned int* (коэффициенты *0* и *1* при числах Фибоначчи должны быть записаны от младшего к старшему слева направо с дополнительной единицей в конце записи, репрезентирующей окончание записи);
    - %Cv печать целого числа типа *int* в системе счисления с заданным основанием (при обработке флага первым параметром функции, “снимаемым” со стека, является целое число типа *int*, вторым - основание целевой системы счисления в диапазоне *[2..36]* (при основании системы счисления, не входящем в диапазон, значение основания системы счисления устанавливается равным *10*)); символы букв в результирующем строковом представлении целого числа должны быть записаны в нижнем регистре;
    - %CV – аналогично флагу %Cv, при этом символы букв во входном строковом представлении целого числа должны быть записаны в верхнем регистре;
    - %to – печать в поток вывода результата перевода целого числа, записанного в строковом представлении в системе счисления с заданным основанием в систему счисления с основанием *10* (при обработке флага первым параметром функции, “снимаемым” со стека, является строка, описываемая значением типа *char **, вторым - основание исходной системы счисления в диапазоне *[2..36]* (при основании системы счисления, не входящем в диапазон, значение основания системы счисления устанавливается равным *10*)); символы букв во входном строковом представлении целого числа должны быть записаны в нижнем регистре;
    - %TO - аналогично флагу %to, при этом символы букв во входном строковом представлении целого числа должны быть записаны в верхнем регистре;
    - %mi – печать дампа памяти (байты значения, записанные в системе счисления с основанием *2*, в порядке нахождения в памяти “слева направо”; строковые представления байтов должны сепарироваться одним символом пробела) значения знакового целого 4-байтного числа;
    - %mu – печать дампа памяти (байты значения, записанные в системе счисления с основанием *2*, в порядке нахождения в памяти “слева направо”; строковые представления байтов должны сепарироваться одним символом пробела) значения беззнакового целого 4-байтного числа;
    - %md – печать дампа памяти (байты значения, записанные в системе счисления с основанием *2*, в порядке нахождения в памяти “слева направо”; строковые представления байтов должны сепарироваться одним символом пробела), значения вещественной переменной типа *double*;
    - %mf – печать дампа памяти (байты значения, записанные в системе счисления с основанием *2*, в порядке нахождения в памяти “слева направо”; строковые представления байтов должны сепарироваться одним символом пробела), значения вещественной переменной типа *float*.

    Продемонстрируйте работу реализованных функций.

10. Реализовать функции *overscanf*, *overfscanf* и *oversscanf*, поведение которых схоже с поведением стандартных функций *scanf*, *fscanf* и *sscanf* соответственно, то есть эти функции имеют одинаковый прототип и логику работы, но в ваших функциях помимо стандартных флагов для функций семейства *scanf* (%d, %u, %f, %lf, %c, %s, %o, %x, %X) добавляются дополнительные флаги:
    - %Ro – считывание из потока ввода целого числа типа *int*, записанного римскими цифрами;
    - %Zr – считывание из потока ввода целого числа типа *unsigned int*, записанного в виде цекендорфова представления (коэффициенты *0* и *1* при числах Фибоначчи должны быть записаны от младшего к старшему слева направо с дополнительной единицей в конце записи, репрезентирующей окончание записи);
    - %Cv считывание из потока ввода целого числа типа *int*, записанного в системе счисления с заданным основанием (при обработке флага первым параметром функции, “снимаемым” со стека, является адрес места в памяти типа *int* *, куда необходимо записать считанное значение, вторым - основание системы счисления (в которой записано число, находящееся в потоке ввода) в диапазоне *[2..36]* (при основании системы счисления, не входящем в диапазон, значение основания системы счисления устанавливается равным *10*)); символы букв во входном строковом представлении целого числа должны быть записаны в нижнем регистре;
    - %CV – аналогично флагу %Cv, при этом символы букв во входном строковом представлении целого числа должны быть записаны в верхнем регистре.

    Валидация данных, находящихся во входном потоке, не требуется.
    Продемонстрируйте работу реализованных функций.

11. Реализовать функцию, выделяющую лексемы из входной строки по предикату. Прототип функции:

    int tokenize(char initial, int (*detector)(int), int accept_empty_lexems, char*** lexems, size_t* lexems_count);*

    Параметры:

    initial - указатель на строку, из которой необходимо выделить лексемы

    detector - указатель на функцию, разделяющую символы, входящие в лексему и являющиеся разделителями:

    возвращаемое значение функции = 0 - символ является разделителем ("плохим")

    otherwise - символ является входящим в лексему ("хорошим")

    accept_empty_lexems - допускать ли в результирующем массиве пустые лексемы:

    0 - пустые лексемы не должны находиться в результирующем массиве лексем

    otherwise - пустые лексемы могут находиться в результирующем массиве лексем

    lexems - указатель, под который необходимо записать массив выделенных лексем

    lexems_count - указатель, под который необходимо записать количество выделенных лексем

    Возвращаемое значение - один из статус-кодов: 0 - функция завершилась успешно; 1 - параметр *initial* имеет значение *NULL*; 2 - параметр *detector* имеет значение *NULL*; 3 - параметр *lexems* имеет значение *NULL*; 4 - параметр *lexems_count* имеет значение *NULL*; 5 - возникла проблема с [пере]выделением памяти во время выполнения функции.
    Для реализации функции используйте принцип границы. Во время работы функции допускается максимум один проход по строке *initial*.
    Добавление символов в лексемы и лексем в коллекцию должно выполняться за O(1) amortized.
    Продемонстрируйте работу функции.

12. На языке программирования C реализуйте функцию поиска подстроки в нескольких строках. Прототип функции:

int substr(char to_find, int case_sensitive, char*** results, int*** positions, size_t* results_count, ...);*

Параметры:

to_find - указатель на строку, которую необходимо найти в строках в качестве подстроки

case_sensitive - флаг, отвечающий за учёт регистра символов букв: 0 - регистр символов букв не учитывается; otherwise - регистр символов букв учитывается

results - указатель на место в памяти, куда необходимо сохранить динамически выделенный массив указателей на строки из списка аргументов переменной длины, в которых нашлась переданная подстрока (согласованно с массивом под указателем *positions*)

positions - указатель на место в памяти, куда необходимо сохранить динамически выделенный массив указателей на массивы, содержащих количество индексов (нулевой элемент) и далее индексы (отсчёт с 0) каждого вхождения подстроки в строку (согласованно с массивом под указателем *results*)

results_count - указатель на место в памяти, куда необходимо сохранить количество строк, в которых нашлась переданная подстрока

... - список аргументов переменной длины, содержащий указатели на строки, в которых необходимо искать переданную подстроку.

Возвращаемое значение - один из статус-кодов: 0 - функция завершилась успешно; 1 - параметр *to_find* имеет значение *NULL*; 2 - параметр *results* является указателем на пустую строку; 3 - параметр *positions* имеет значение NULL; 4 - параметр *results_count* имеет значение *NULL*; 5 - возникла проблема с [пере]выделением памяти во время выполнения функции.

Указатель со значением *NULL*, передаваемый в список аргументов переменной длины, является последним обрабатываемым аргументом в списке аргументов переменной длины.

Результирующий массив строк должен содержать копии исходных строк (необходимо глубокое копирование, а не поверхностное).

Продемонстрируйте работу функции.

13. На языке программирования C реализуйте функцию, находящую все возможные (с точностью до порядка следования) уникальные разложения натурального числа на суммы натуральных чисел. Прототип функции:

*int sums_decomposition(int value, int *** result_decompositions, size_t *result_decompositions_count, int allowed_equal_sum_components);*

Параметр *allowed_equal_sum_components* влияет на то, могут ли компоненты любого разложения числа *value* на суммы натуральных чисел повторяться (значение отлично от 0) или нет (значение равно 0).

Результирующую коллекцию разложений натурального числа *value*, размещённую в динамической памяти, необходимо вернуть из функции через параметр *result_decompositions*; их количество - через параметр *result_decompositions_count*. На уровне каждого разложения, репрезентируемого коллекцией вида динамический массив: его нулевой элемент задаёт количество компонентов, участвующих в разложении, первый и далее элементы - сами компоненты разложения.

Возвращаемое значение - один из статус-кодов: 0 - функция завершилась успешно; 1 - параметр *result_decompositions* имеет значение *NULL*; 2 - параметр *result_decompositions_count* имеет значение *NULL*; 3 - значение *value* не является натуральным числом; 4 - возникла проблема с [пере]выделением памяти во время выполнения функции.

Продемонстрируйте работу функции.

14. На языке программирования C реализуйте функцию, находящую в массиве элементов типа *tvalue* наибольшую по переданному компаратору пилообразную подпоследовательность. Прототип функции:

*int find_the_longest_sawtooth_subsequence(tvalue const *sequence, size_t *subsequence_start_index_storage, size_t *subsequence_length_storage, int (*comparer)(tvalue const *, tvalue const *), int is_comparison_is_strict);*

Пилообразная подпоследовательность - подпоследовательность исходной последовательности элементов, такая, что каждый её элемент либо больше своих существующих соседей, либо меньше своих существующих соседей.

Если наибольшая пилообразная подпоследовательность найдена, то индекс её начального элемента необходимо сохранить под указатель *subsequence_start_index_storage*, длину этой подпоследовательности необходимо сохранить под указатель *subsequence_length_storage*; в противном случае под указатели необходимо сохранить числа *-1* и *0* соответственно.

Сравнение элементов последовательности на предмет отношения порядка выполняется посредством параметра *comparer*. Параметр *is_comparison_is_strict* задаёт строгость/нестрогость сравнения на предмет отношения порядка: *0* - сравнение нестрогое; otherwise - строгое.

Возвращаемое значение - один из статус-кодов: 0 - функция завершилась успешно; 1 - параметр *sequence* имеет значение *NULL*; 2 - параметр *subsequence_start_index_storage* имеет значение *NULL*; 3 - параметр *subsequence_length_storage* имеет значение *NULL;* 4 - параметр *comparer* имеет значение *NULL*.

Обработайте всевозможные ошибки, которые могут возникнуть во время работы функции. Продемонстрируйте работу функции.

15. На языке программирования C реализуйте функцию, находящую в матрице значений типа *tvalue* все седловые точки. Прототип функции:

*int find_saddle_points(tvalue const * const *matrix, size_t matrix_rows_count, size_t matrix_columns_count, size_t ***found_saddle_points_storage, size_t *found_saddle_points_count_storage, int (*comparer)(tvalue const *, tvalue const *), int is_comparison_is_strict);*

Седловая точка матрицы - значение, являющееся наименьшим в своей строке и наибольшим в своём столбце.

Координаты найденных седловых точек необходимо сохранить в динамический массив под указатель *found_saddle_points_storage*, количество найденных точек - под указатель *found_saddle_points_count_storage.*

Сравнение элементов последовательности на предмет отношения порядка выполняется посредством параметра *comparer*. Параметр *is_comparison_is_strict* задаёт строгость/нестрогость сравнения на предмет отношения порядка: *0* - сравнение нестрогое; otherwise - строгое.

Возвращаемое значение - один из статус-кодов: 0 - функция завершилась успешно; 1 - параметр *matrix* имеет значение *NULL*; 2 - указатель на любую из строк матрицы *matrix* имеет значение *NULL*; 3 - параметр *found_saddle_points_storage* имеет значение *NULL*; 4 - параметр *found_saddle_points_count_storage* имеет значение *NULL;* 5 - параметр *comparer* имеет значение *NULL;* 6 - возникла проблема с [пере]выделением памяти во время выполнения функции..

Обработайте всевозможные ошибки, которые могут возникнуть во время работы функции. Продемонстрируйте работу функции. После завершения демонстрации работы функции необходимо освободить всю выделенную в её контексте динамическую память. При/после работы функции недопустимы потенциальные утечки памяти.

16. На языке программирования C реализуйте функцию, находящую все возможные уникальные перестановки элементов входной коллекции типа массив. Прототип функции:

*int permutations(int *items, size_t items_count, int ** result_permutations, size_t *result_permutations_count, int (*equality_comparer)(int const *, int const *));*

Значения коллекции *items* необходимо проверить на уникальность (любые два значения в коллекции должны быть различны по переданному отношению эквивалентности *equality_comparer*; в случае, если это условие не выполняется, вычисление коллекции перестановок производить не нужно).

Результирующую коллекцию перестановок, размещённую в динамической памяти, необходимо вернуть из функции через параметр *result_permutations*; их количество - через параметр *result_permutations_count*. При реализации функции **запрещено** использовать стандартную функцию *realloc*.

Возвращаемое значение - один из статус-кодов: 0 - функция завершилась успешно; 1 - параметр *items* имеет значение *NULL*; 2 - параметр *result_permutations* имеет значение *NULL*; 3 - параметр *result_permutations_count* имеет значение *NULL*; 4 - параметр *equality_comparer* имеет значение *NULL*; 5 - возникла проблема с выделением памяти во время выполнения функции; 6 - в коллекции найдены элементы, равные по переданному в функцию отношению эквивалентности.

Обработайте всевозможные ошибки, которые могут возникнуть во время работы функции. Продемонстрируйте работу функции. После завершения демонстрации работы функции необходимо освободить всю выделенную в её контексте динамическую память. При/после работы функции недопустимы потенциальные утечки памяти.

17. На языке программирования C реализуйте функцию, находящую множество всех уникальных подмножеств для элементов входной коллекции типа массив. Прототип функции:

*int subsets(int *items, size_t items_count, int ** result_subsets, size_t *result_subsets_count, int (*equality_comparer)(int const *, int const *));*

Значения коллекции *items* необходимо проверить на уникальность (любые два значения в коллекции должны быть различны по переданному отношению эквивалентности *equality_comparer*; в случае, если это условие не выполняется, вычисление множества всех подмножеств производить не нужно).

Результирующую коллекцию подмножеств, размещённую в динамической памяти, необходимо вернуть из функции через параметр *result_subsets* (для каждого подмножества, нулевой элемент результирующего массива - количество элементов в подмножестве, первый и далее - элементы подмножества); их количество - через параметр *result_subsets_count*. При реализации функции **запрещено** использовать стандартную функцию *realloc*.

Возвращаемое значение - один из статус-кодов: 0 - функция завершилась успешно; 1 - параметр items имеет значение *NULL*; 2 - параметр *result_subsets* имеет значение *NULL*; 3 - параметр *result_subsets_count* имеет значение *NULL*; 4 - параметр *equality_comparer* имеет значение *NULL*; 5 - возникла проблема с выделением памяти во время выполнения функции; 6 - в коллекции найдены элементы, равные по переданному в функцию отношению эквивалентности.

Обработайте всевозможные ошибки, которые могут возникнуть во время работы функции. Продемонстрируйте работу функции. После завершения демонстрации работы функции необходимо освободить всю выделенную в её контексте динамическую память. При/после работы функции недопустимы потенциальные утечки памяти.

18. На языке программирования C реализуйте функцию, находящую все возможные уникальные сочетания по *k* элементов для элементов входной коллекции типа массив. Прототип функции:

*int combinations(int *items, size_t items_count, int ** result_combinations, size_t *result_combinations_count, int (*equality_comparer)(int const *, int const *), size_t k);*

Значения коллекции *items* необходимо проверить на уникальность (любые два значения в коллекции должны быть различны по переданному отношению эквивалентности *equality_comparer*; в случае, если это условие не выполняется, вычисление множества всех подмножеств производить не нужно).

Результирующую коллекцию сочетаний по *k* элементов из *items_count* элементов, размещённую в динамической памяти, необходимо вернуть из функции через параметр *result_subsets*; их количество - через параметр *result_subsets_count*. При реализации функции **запрещено** использовать стандартную функцию *realloc*.

Возвращаемое значение - один из статус-кодов: 0 - функция завершилась успешно; 1 - параметр items имеет значение *NULL*; 2 - параметр *result_combinations* имеет значение *NULL*; 3 - параметр *result_combinations_count* имеет значение *NULL*; 4 - параметр *equality_comparer* имеет значение *NULL*; 5 - значение *k* больше, чем значение *items_count*; 6 - возникла проблема с выделением памяти во время выполнения функции; 7 - в коллекции найдены элементы, равные по переданному в функцию отношению эквивалентности.

Обработайте всевозможные ошибки, которые могут возникнуть во время работы функции. Продемонстрируйте работу функции. После завершения демонстрации работы функции необходимо освободить всю выделенную в её контексте динамическую память. При/после работы функции недопустимы потенциальные утечки памяти.

19. На языке программирования C реализуйте функцию, находящую все возможные конфигурации башен из кубиков. Прототип функции:

*int towers_construction(int blocks_count, int ** result_towers, size_t *result_towers_count, int allowed_partial_blocks_usage, int allowed_adjacent_layers_blocks_equal_count);*

Параметр *allowed_partial_blocks_usage* влияет на то, необходимо ли для построения башни использовать обязательно все *blocks_count* кубиков (значение отлично от 0) или нет (значение равно 0).

Параметр *allowed_adjacent_layers_blocks_equal_count* влияет на возможное количество кубиков в двух соседних слоях башни: они могут быть одинаковы (значение отлично от 0) или же нет (значение равно 0). При этом, независимо от значения параметра, количество кубиков на вышележащем слое не может превышать количество кубиков на нижележащем.

Результирующую коллекцию слоёв (задаётся количеством кубиков) башен (от нижележащего к вышележащему), размещённую в динамической памяти, необходимо вернуть из функции через параметр *result_towers* (для каждой конфигурации, нулевой элемент результирующего массива - количество слоёв в башне, первый и далее - количество кубиков в слоях от нижележащего к вышележащему); их количество - через параметр *result_towers_count*.

Возвращаемое значение - один из статус-кодов: 0 - функция завершилась успешно; 1 - параметр *result_towers* имеет значение *NULL*; 2 - параметр *result_towers_count* имеет значение *NULL*; 3 - значение *blocks_count* является отрицательным числом; 4 - возникла проблема с [пере]выделением памяти во время выполнения функции.

Обработайте всевозможные ошибки, которые могут возникнуть во время работы функции. Продемонстрируйте работу функции. После завершения демонстрации работы функции необходимо освободить всю выделенную в её контексте динамическую память. При/после работы функции недопустимы потенциальные утечки памяти.