# Подготовка проекта

Настройка CMake и автоматического тестирования Catch2.

CMake – система сборки проектов для C++.

* Умеет создавать проекты для разных сред и компиляторов по Windows, Linux, MacOS из универсальных файлов
* Умеет искать и подключать библиотеки из стандартных и указанных папок
* Поддерживается средами разработки (Visual Studio, Qt Creator, Visual Studio Code)

Catch2 – библиотека для модульного тестирования (Unit Tests).

* Умеет вместо одного метода main добавить в программу много проверочных функций (тестов), которые могут завершиться успешно или нет
* Для подключения этой библиотеки, в отличие от аналогичных Google Test и Boost Test, достаточно подключения одного заголовочного файла.
* Если один из тестов завершается с ошибкой, пользователь или получают читаемое сообщение типа «эта функция должна была вернуть 5, а вернула 10»
* Модульные тесты используют те же h-файлы и .cpp, что и основная программа и позволяют автоматически проверять работоспособность кода
* Тесты могут объединяться в группы, запускаться по одному и так далее.
* Тестовое приложение – отдельный .exe файл с множеством проверок и отдельными опциями запуска.

<https://ps-group.github.io/cg/mmz01>  
<https://github.com/catchorg/Catch2/blob/master/docs/tutorial.md#top>

1. Создайте папку для проекта
2. Разместите в ней файлы Catch.hpp <https://github.com/catchorg/Catch2/releases/download/v2.11.1/catch.hpp>

и заготовки файлов программы

<https://yadi.sk/d/bqmWy8o0OexDgA>

1. Чтобы в проекте компилировались main.exe и test.exe, создайте текстовый файл

**CMakeLists.txt** следующего содержания:

project(lab1-asd)

cmake\_minimum\_required(VERSION 3.12)

option(BUILD\_TESTS "Build test" ON)

add\_executable(main main.cpp generator.cpp generator.h catch.hpp)   
# компилируем main.exe из cpp и h файлов (h,hpp для проверки изменений по дате файлов)

if(BUILD\_TESTS)

enable\_testing()

add\_executable(test1 test.cpp generator.cpp generator.h catch.hpp)   
 # компилируем test1.exe

add\_test(test1 test1)   
#запускаем test1.exe как тестовую программу. Тест пройден, если она возвращает код 0.

endif()

1. Откройте папку в MS Visual Studio (2019) – File – Open Folder. Дождитесь настройки проекта (ошибок в выводе быть не должно). Выберите конфигурацию x64-Debug и нужный исполняемый файл



Создайте также конфигурацию x64 – Release (Manage Configurations в списке конфигураций).

1. Запустите test1.exe и внимательно просмотрите информацию о пройденных и не пройденных тестах. Исправьте ошибку, чтобы все 10 проверок в 2 тестах прошли успешно.
2. Создайте в классе Generator поле для хранения некого набора последовательностей чисел типа vector<vector<int>>
3. Добавьте в методе GenerateAll() три элемента к этому набору (push\_back), например, {1,2,3}, {1,2,4}, {1,3,4}.
4. Организуйте получение последовательности по номеру в методе GetResult
5. Добавьте к Test.cpp тест (новый TEST\_CASE) с проверками (REQUIRE) правильности первых трёх ответов (GetResult(0), GetResult(1), GetResult(2)).

# Лабораторная работа №1

Написать программу, реализующую алгоритм построения всех последовательностей длины n некоторого вида.

* Оформить в виде класса. Текущую последовательность хранить в поле (массивом или вектором)
* Передавать n и k рекомендуется в конструктор класса
  + Получаемые последовательности печатать на экран, если в классе установлена опция (поле) printFlag=true (сделать параметр конструктора или метод установки)
  + Рекомендуется создать отдельную функцию печати последовательности
* Получаемые последовательности подсчитывать (всегда) и заносить в список, если установлена переменная saveFlag (сделать свойство или метод установки)
  + Чтобы занести массив в список, проще всего использовать тип vector C++ :

vector<vector<int>> data

…

data.push\_back( vector<int>(a,a+n) )

(создаёт новый vector<int> из фрагмента масcива и добавляет в конец списка)

* Автоматическими тестами проверить для фиксированного n, k
  + правильность количества построенных последовательностей
  + содержимое последовательности с заданным номером
* Требование: последовательности, не удовлетворяющие условиям, не должны появляться при построении (а не просто игнорируются при печати и сохранении)

Что генерировать (все в лексикографическом порядке, кроме кода Грея):

Пробный вариант: все последовательности длины n с элементами 1..k в лексикографическом порядке:

111, 112, 113, 121 ….

* Сначала на бумаге или в комментариях к коду пишется искомая последовательность при небольших n и k
* Формулируется правило «x[j] можно увеличить, не меняя предыдущих элементов тогда и только тогда, …» для поиска j (без рекурсии) или определения конечного значения в цикле (с рекурсией)
* Для способа без рекурсии – составляется алгоритм заполнения хвоста x[j+1], x[j+2],… после увеличения x[j]
* Для способа с рекурсией – выбирается начальное значение для цикла (может зависеть от предыдущего элмента или нескольких, текущей суммы, текущего количества нулей и единиц и т.д.)

Каждому студенту выбрать 1 из «легких» вариантов, не более 2 одинаковых в группе, сделать без рекурсии и с рекурсией.

1. Сочетания предметов (возрастающие последовательности номеров)
2. Последовательности из нулей и единиц длины n, в которых ровно k единиц.
3. Перестановки n элементов в лексикографическом порядке.
4. Сочетания предметов с повторениями (неубывающие последовательности длины n с элементами 1..k типа 122,123,133,222,…)
5. Последовательности из нулей и единиц длины n, в которых от k1 до k2 единиц.
6. Разбиения числа m на n слагаемых 1..m (произвольная последовательность)

Каждому студенту выбрать 1 из «сложных» вариантов, без повторов в группе, сделать без рекурсии и с рекурсией.

1. Разбиения числа m на n слагаемых 1..m (неубывающая последовательность)
2. Разбиения числа m на n слагаемых от 1..k (неубывающая последовательность)
3. Разбиения числа m на n слагаемых от 1..k (произвольная последовательность)
4. Сочетания предметов, где каждый предмет можно брать не более 2 раз (неубывающие последовательности длины n с элементами 1..k типа 122,123,133, но 222 нельзя)
5. Последовательности нулей и единиц, в которых ровно k единиц и нули идут минимум по 3 шт подряд.
6. Последовательности нулей и единиц, в которых ровно k единиц и встречаются не более 3 единиц подряд.
7. Разбиения числа m на n слагаемых от 1..k (возрастающая последовательность)
8. Разбиение числа m на произвольное число слагаемых от 1 до k (произвольная последовательность)
9. Разбиение числа m на произвольное число слагаемых от 1 до k (возрастающая последовательность)
10. Разбиение числа m на произвольные неубывающие множители больше 1 (например, для 24 построить 2,2,2,3 2,2,6 2,3,4 3,8 4,6 )
11. Последовательности нулей и единиц, в которых не более k единиц и встречаются не более 3 единиц подряд.
12. Последовательности нулей и единиц, в которых не менее k единиц и нули идут минимум по 2 шт подряд.
13. // Lab\_1.cpp : This file contains the 'main' function. Program execution begins and ends there.
14. //
15. #include <iostream>
16. #include <vector>
17. #include <algorithm>
18. using namespace std;
19. vector <int> temp(1, 1);
20. vector <int> original;
21. vector <vector<int>> result;
22. void initializeArray(int number)
23. {
24. int summ = 1;
25. int var = 2;
26. int counter = 0;
27. while (summ != number)
28. {
29. if (summ + var > number)
30. {
31. while (summ - temp[counter] + var != number)
32. var++;
33. temp[counter] = var;
34. break;
35. }
36. temp.push\_back(var);
37. summ += var;
38. counter++;
39. var++;
40. }
41. }
42. int main()
43. {
44. int originalNumber;
45. cin >> originalNumber;
46. initializeArray(originalNumber);
47. for (int i = 0; i < temp.size(); i++)
48. cout << temp[i] << " ";
49. return 0;
50. }