Home task

1.Частичная специализация шаблона (partial template specialization)

Необходимо воспользоваться заготовкой template_of_HW.cpp на гитхабе в папке CppLessons/29/hw

```
template<class T, size_t N>
class Container{
    private:
        T m_array[N];
    public:

        Container(){
            for(int i = 0 ; i < N ; i ++)
            {
                m_array[i] = (T)0;
            }
        }

        void print(){

            for(const auto & a: m_array)
            {
                std::cout<<a<<"\t";
            }
            std::cout<<"\n";
        }

        void fill(const T array[N])
        {
            std::copy(array,array + N , m_array);
        }


};
```

Это шаблонный класс с двумя параметрами – параметр класса и non-type параметр. Необходимо частично его специализировать:

1.1.

```cpp
class Item
{
    private:
        int * m_arr;
        size_t m_size;
    public:
    Item();
    Item(size_t size);//необходимо выделить сюда массив размера size и заполнить его рандомными числами до 100
    //как делать --- см rand()

    friend std::ostream & operator<<(std::ostream& out, const Item & it );
    double average() const;//не забываем о приведении типов.

    ~Item();


};

template<size_t N>
class Container<std::vector<Item>, N >
{
    private:
        std::vector<Item> m_vec;
    public:
        Container();
        void add(const Item & it);//должна быть проверка, что
        //кол-вао элементов <= N

        void print(); // для всех обьектов, которые содержатся в контейнере
        // m_vec необходимо вывести среднее арифметическое
};
```

Необходимо определить объявленные методы класса Item согласно краткому описанию в комментариях. Функция **double average() const** должна возвращать среднее арифметическое массива m_arr.

Что касается шаблонного класса **Container,** его необходимо специализировать таким образом, как указано на скрине. Поле данных изменяется на **std::vector<Item> m_vec;** , методы, соответственно, тоже поддаются изменениям. Конструктор должен резервировать память ровно под N объектов. (у вектора есть метод reserve()). А функция print() должна печатать среднее арифметическое для каждого объекта Item внутри этого вектора. Функция **void add(…)** , как бы шокирующе это не звучало, должна добавлять обьект типа **Item** в **m_vec.**

1.2.

```cpp
template<size_t N>
class Container<std::list<std::string>, N>
{
    private:
        std::list<std::string> listOfStr;
    public:
        Container();//здесь ничего резервировать не надо
        void add(const std::string & str); // не забываем проверку

        void print();//для каждой строки, которая содержится в контейнере list
        //необходимо вывести в консоль количество запятых в строке

};
```

Определить методы специализации при входном параметре шаблона **std::list<std::string>** . Функция **void print()** должна печатать для каждого объекта типа **std::string** внутри нашего контейнера - количество запятых. **void add(…) –** добавляет строку внутрь нашего контейнера **std::list.**

!!!Все объявленные функции необходимо определить, даже если они не применяются. Внутри класса или вне – решать вам!!!

Работу обеих специализаций отобразить на примерах в мейне.

2. STL containers

На следующем занятии будет тест(несложный) по последовательным контейнерам (sequence containers). Поэтому вам необходимо будет отработать и запомнить как устроены
-вставка (в любое место контейнера)
-удаление
-обход (по итератору в том числе)
для контейнеров : list, vector, array, deque.