Genesis Roman
Group #5

8. **Create a *templated class* that *effectively* finds all possibilities of a list of random numbers that adds to some s.**

In this program we will see in the output a list of random numbers which were picked for the server following all the steps that I as programmer made to find the possibilities in a template class. All the STL name are part of the std namespace, which means you must either use the characters std:: before each and every STL, but if you do not include “using namespace” statement, items from the template must be qualified with the std:: prefix.

In this program I declared all the variables so it will be base in a range of min and max number. Random class is made with all the instruction so server can process randomly numbers.

```
#include <iostream>
#include <math.h>
#include <memory>
#include <random>
#include <string>
#include <vector>

// for some compilers that's don't support nested templates with the same parameter
template <typename T>
auto dist() -> typename std::enable_if<std::is_integral<T>::value,
std::uniform_int_distribution<T> >::type;

template <typename T>
auto dist() -> typename std::enable_if<std::is_floating_point<T>::value,
std::uniform_real_distribution<T> >::type;

template<typename T>
class Random
{
public:
   Random(const T& min, const T& max)
      : mUnifomDistribution(min, max)
   {}
   Random(const Random<T>&) = delete;
   Random(const Random<T>&&) = delete;
   Random<T>& operator = (const Random<T>&) = delete;
   T operator()()
   {
      return mUnifomDistribution(mEngine);
   }

private:
   std::default_random_engine mEngine{ std::random_device()() }; // <-- here the doubt - is it
seeding?
```

Genesis Roman
Group #5

```
    //template <typename T>
    //static auto dist() -> typename std::enable_if<std::is_integral<T>::value,
std::uniform_int_distribution<T> >::type;

    //template <typename T>
    //static auto dist() -> typename std::enable_if<std::is_floating_point<T>::value,
std::uniform_real_distribution<T> >::type;

    using type = decltype(dist<T>());
    type mUnifomDistribution;
};

int main()
{
    ::Random<int> s(0, 9);
    for (int i = 0; i<9; ++i)
        std::cout << s() << '\n';
}
```