Effective and Modern C++ Programming

Lab 14 – Metaprogramming

Exercise 1. Recurrent classes

Implement templates that compute during compilation:

- n-th power of given integer number,
- binomial coefficient $\binom{n}{k} = \frac{n!}{k!(n-k)!} = \binom{n-1}{k} + \binom{n-1}{k-1}$

```
std::cout << Power<5,3>::value;   // 125

std::cout << Binomial<4,2>::value << std::endl; //6
std::cout << Binomial<100,0>::value << std::endl; //1
std::cout << Binomial<100,1>::value << std::endl; //100
std::cout << Binomial<100,7>::value << std::endl; //16007560800
```

Exercise 2. constexpr, consteval & constinit

a) Implement function

consteval auto power(auto x, int n)

that computes x^n using the following recurrent formula
$x^{2n+1} = x^{2x} * x$

$x^{2n} = (x^n)^2$

b) Implement function generatePascalTriangle<N> that during compilation generates the PascalTriangle<N> of the given size N with precomputed Binomial coefficients.Pascal triangle: https://en.wikipedia.org/wiki/Pascal%27s_triangle

constinit double fiveToPowerFour = power(5, 4);

constexpr size_t n = 10;

constexpr auto triangle = generatePascalTriangle<n>();

static_assert(triangle(0,0) == 1);

static_assert(triangle(5,3) == 10);

static_assert(triangle(9,4) == 126);

Exercise 3. IntegerList

Implement

- variadic class template IntegerList, that can “store” integer values as template arguments,
- template class getInt<index, IntegerList> that turns element with a given index for the IntegerList,
- template class Join<IntegerList1, IntegerList2> that joins two IntegerLists,
- template class IsSorted<IntegerList> that checks if arguments of IntegerList are sorted,
- template class Max<IntegerList> that finds maximal element in given list of integers.

In class IntegerList implement static constexpr functions:

- get(int index) that returns with a given index for the IntegerList,
- max() that finds maximal element in the IntegerList.

```
int main() {

    using listA = IntegerList<5, -1, 5, 2, 1>;
    using listB = IntegerList<1, 4, 6, 9>;
    cout << "List A : " << listA() << endl;
    cout << "List B : " << listB() << endl;
    cout << getInt<1, listA>::value << endl;
    cout << listB::get(3) << endl;

    using listC = Join<listA, listB>::type;
    cout << "List C : " << listC() << endl;

    cout << boolalpha;
    cout << "Is A sorted? " << IsSorted<listA>::value << endl;
    cout << "Is B sorted? " << IsSorted<listB>::value << endl;

    cout << Max<listC>::value << endl;
    cout << listC::max() << endl;
}
```