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. Expressions unforlding

Implement templates that unfolds the following operations:

- ScalarProduct<N, T>(a, b) - scalar product of two vectors a and b, (N is a size of vectors, T is type of its elements)
- Multiply<N, M, T>(A, x , y) – multiplies matrix A by vector x and stores result in y: y=Ax (N, M are sizes of a matrix, T is type of its elements)

There should be no loops in the code. The expression should be unfolded so that it can be computed during compilation.

```
double a[] = {1, 2, 3};
double b[] = {1, 1, 1};
std::cout<< ScalarProduct<3>(a, b); // 6

double x[] = {1, 1, 0};
double A[] = {1, 0, 0,
              2, -5, 1};
double y[2];
Multiply<2,3>(A, x, y);
std::cout << y[0] << " " << y[1];  // 1 -3
```

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;
}
```