Lab 4 - Function templates

Exercise 1.

A) Implement a function template **sqr** with one type template parameter T

```
T sqr(T x)
```

that returns x*x for all numeric types.

Define specialization for std::string that returns doubled string e.g for “ab” it returns “abab”

```
cout << sqr(4) << endl;              // 16
cout << sqr(14.5) << endl;           // 210.25
cout << sqr(string("hey")) << endl;  // heyhey
```

B) Implement a function template mod with one non-type template parameter N (of type int)

```
int mod<N>(int x)
```

that returns x modulo N. Specialization for N=0 should return -x;

```
cout << mod<5>(131) << endl;         // 1
cout << mod<7>(131) << endl;         // 5
cout << mod<0>(131) << endl;         // -131
```

C) Implement a function template

```
void print(const Container & v)
```

that prints to standard output all elements of the given container separating elements by a single space.

```
std::vector<int> v = {1, 21, 34, 4, 15};
print(v);                    // 1 21 34 4 15

std::list<double> l = {1, 2.1, 3.2, 6.3};
print(l);                    // 1 2.1 3.2 6.3
```

D) Implement a function template apply with two template parameters

C – type of container

F – type of function or functional object

```
C apply (const C& c, F f)
```

that for each element x in c calls f(x) and inserts returned result to new container. Function should return this new container (of type C).

```
auto w = apply(v, sqr<int>);
auto w2 = apply(w, mod<5> );
print(w2); // 1 1 1 1 0

auto l2 = apply(l, sqr<double>);
auto l3 = apply(l2, mod<5>);
print(l3); // 1 4 0 4
// function sin is overloaded, we need to cast it
auto l4 = apply(l3, static_cast<double(*)(double)>(std::sin));
print(l4); // 0.841471 -0.756802 0 -0.756802
```

Exercise 2.

Implement a function template

```
int compare(T a, T b)
```

Template function should return:

- 1 if a < b,
- -1 if b < a,
- 0 otherwise

We assume only that objects a and b are comparable using operator <.
In particular it should work for all integer and floating point types.

Implement the **specializations of the function** template compare:

- for pointers: it should compare pointed objects instead of pointers itself,
- for pointers to C strings: it should compare strings lexicographically i.e. ”call” < ”car”

```
int a = 1, b=-6;
float  y= 1.0 + 1e20 - 1e20, x = 1.0;
cout << compare(a,b) << " " << compare(b,a) << " " << compare(a,a) << endl;
cout << compare(x,y) << " " << compare(y,x) << " " << compare(x,x) << endl;
cout << compare(&a,&b) << " " << compare(&b,&a) << " " << compare(&a,&a) <<endl;
cout << compare(&x,&y) << " " << compare(&y,&x) << " " << compare(&x,&x) <<endl;
cout << compare(“Alpha”, “Alfa”) <<endl;
OUTPUT
-1 1 0
-1 1 0
-1 1 0
-1 1 0
-1
```

Exercise 3.

Implement a function template *process* that has three template parameters

- T – the type of array elements
- f – a pointer to function with one argument of type T and return type T
- N – the number of elements in the array

```
void process(T array[]);
```

The function *process* for each element in given array calls function f and replaces this element with the result of the call.

```
double a[] = {1, 2, 3, 4};
process<double, sin, 4> (a);
for( auto x: a)
      cout << x << " "; // 0.841471 0.909297 0.14112 -0.756802
```

Exercise 4.

Implement a function template

OutContainer<T,Alloc> selectIf(InContainer<T,Alloc> c, Predicate p);

It should return container that contains all elements from container c for which predicate p returns true.
Template parameters are

- OutContainer<T, Alloc> - template with two parameters that
- T – the type of the elements in the container
- Alloc – the type of the allocator
- InContainer<T, Alloc> – template with two parameters
- Predicate – the type of function or functional object that takes one argument of type T and returns bool.

OutContainer, InContainer can be any of standard sequence containers e.g. vector, list, deque.

```
bool biggerThan5(int x){ return x>5; }
...
std::vector<int> v={1, 2, 13, 4, 5, 54};
std::list<int> result = selectIf<std::list>(v, biggerThan5);
// result should contain 13 and 54
```