Sarah Naja
Felix Becker
Philipp Hertkorn
Yunus Tan

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Programming in C++

Sheet 7

Submission date: 10.10.2024 12:00

7.1 Template Functions (60P = 5P + 5P + 5P + 5P + 20P + 20P)

C++

Now, let's engage in practicing the implementation of template functions. Each subtask in this sheet includes example code illustrating the appropriate function invocation. For additional examples, please refer to the commented out prints in `main.cpp` of `student_template_7.1`.
Please implement the following functions in the given `submission/exercise_07.h` header file.

a) A function `square` that computes the square of the given number of arbitrary type. The return value should have the same type as the input value.

```
int result = square<int>(5);
```

b) A function `halve` that returns halve of the given number of arbitrary type. The return value should always have type `double`.

```
double result = halve<int>(5);
```

c) A function `add` that adds two numbers of the same type together and returns the result in the same type.

```
int result = add<int>(40, 2)
```

d) A function `multiply` that multiplies two numbers of the same type together and returns the result in the same type.

```
int result = multiply<int>(40, 2)
```

e) A function `reduce` that accepts a template operator function of type `std::function`, that combines two arguments of the same type, such as `add` or `multiply`, a `std::vector` of values of the same type and a neutral argument with respect to the operator. Starting from the neutral argument `reduce` iterates over the input `std::vector` and element wise reduces it to one number using the operator function. For example if we use `add` as operator and `0` as neutral element `reduce` will compute the sum like this:

```
std::vector<int> int_data = {1, 2, 3};
int sum_int = reduce<int>(add<int>, int_data, 0)
```

f) A function `map` that applies a given unary template function such as `square` or `halve` to each element in a given `std::vector` and returns the modified elements in a new `std::vector<T>`. The function is provided as `std::function`.

```
std::vector<int> int_data = {1, 2, 3};
std::vector<int> result = map<int>(square<int>, int_data);
```

7.2 Template Class (40P = 10P + 15P + 15P)

C++

We have been using template classes like e.g. `std::vector<T>` for a while now. In this task you will create a simple template class that represents a ComplexNumber.

a) In the provided header file `submission/ComplexNumber.h` we already implemented the `class ComplexNumber` with the member fields `double real_number` and `double imaginary_number`. We want to have a template class instead, allowing arbitrary types. Change the provided class to allow any type, e.g.:

```
ComplexNumber<int> cn = ComplexNumber(1, 2);
```

b) Complete the unary `operator+=` and the binary `operator+`.

```
ComplexNumber<int> cn = ComplexNumber<int>(1, 2) + ComplexNumber<int>(3, 4);
cn += ComplexNumber<int>(3, 4);
```

c) Complete the overloading of the `<< operator` in order to print a `ComplexNumber` to the output stream.

```
std::cout << ComplexNumber<int>(1, 2) << std::endl;
>> 1+2i
```