ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Programming Techniques for Scientific Simulations Exercise 6

HS 15
Prof. M. Troyer

Problem 6.1 Operator overloading and template functions

This exercise focuses on the implementation of a new type describing the $\mathbb{Z}_2$ group, and on the implementation of a generic power function working with all standard numeric types, as well as our new $\mathbb{Z}_2$.

Definition The finite group $\mathbb{Z}_2$ has the following properties:

- Group's element are: $\mathbb{Z}_2 = \{+, -\}$, where $+$ is the identity element.
- The group operation $\cdot$ is defined through:

$$+ \cdot + = - \cdot - = +$$
$$- \cdot + = + \cdot - = -$$

- The representation of a group element $g(\mu)$, $\mu \in \mathbb{Z}_2$ on integer, real or complex numbers is given by

$$g(\mu) = \begin{cases} +1, & \text{for } \mu = + \\ -1, & \text{for } \mu = - \end{cases}$$

a) Implementation of $\mathbb{Z}_2$ To represent the $\mathbb{Z}_2$ group in C++, we will use the enumeration type.

```
enum Z2 { Plus, Minus };
```

The group operation will be implemented by overloading the * operator, i.e. we assign the correct meaning to the expression

```
Z2 p = Plus, m = Minus;
Z2 r = p*m;
```

Every time we make use of the operator *, the C++ compiler is looking for the function `operator*` with the correct types to be invoked[1]. In our case, we need to define:

```
Z2 operator*(Z2 a, Z2 b);
```

Furthermore, we want to be able to print our result in a nice form, i.e. using an expression such as `std::cout << r << std::endl;`. We will therefore overload

```
ostream& operator<<(ostream& os, Z2 a);
```

in such a way that *Plus* is printed for $+$ and *Minus* for $-$.
To implement the action of a group element on a number, we will implement the following template function (note that from the point of view of C++, `a*b` is not necessarily the same as `b*a`):

```
template<class T> T operator*(T a, Z2 b);
template<class T> T operator*(Z2 a, T b);
```

[1] The same is also valid for all operators: $+, -, (), [], <<$, etc.

b) Implementation of generic power function We also want to implement a templated power function which only relies on the multiplication, so that it can also be used on our $\mathbb{Z}_2$ group:

```
template<class T> T mypow(T a, unsigned int n);
```

Hint: in order to be generic we provide a templated function `identity_element`, which returns one for all numeric types. How can you overload such a function, in order provide the identity element of the $\mathbb{Z}_2$ group?

if func obj -> use defined types

if func pointer : template<classR, classT
struct domain_type<R(*)(T)> { ... }

Problem 6.2 Type traits

In a previous exercise we have written function performing simpson integration using function object approach. The solutions in the repository may be called with function pointers as well, as they meet the concept requirement on function objects.[2]
In the previous exercise we pointed out that the templated boundaries are a potential issue, if the type for the boundaries does not meet the required concept, e.g. if they were `int`s. In order to fix that we are going to use traits.
The integrate routine shall be templated only on the function object `F`, the boundary types and the result type shall be deduced from the type `F` via traits

Asssume that F implements result type and argument type

- `domain_type<F>` defining the type for the boundaries,
- `result_type<F>` defining the return type for the integrate function.

The general versions of the traits shall assume existence of `argument_type`, and `result_type` member of the class `F`. Trait `domain_type` shall define the `type` according to the `argument_type` of `F`, and the trait `result_type` shall define the `F`'s member `result_type` as the `type`.
Both traits class needs to be specialized for function pointers with type `R(T)`, and the `R` should be used by the trait `result_type` and the `T` shall be used by the `domain_type`.

Problem 6.3 Penna Model Implementation

In the lecture repository you find the files `genome.hpp` and `animal.hpp` for the solution of the previous exercise. Please refer to these headers for the next parts of the Penna model exercise.

1. Implement all the functions mentioned in the class headers.
2. Test the functionality of your implementation using unit tests.

[2] http://en.cppreference.com/w/cpp/concept/FunctionObject