UNIVERSITÉ DU LUXEMBOURG

Functional Programming in C++

Parallel Computing

Goals

- ✯ Learn C++ lambda.
- ✯ Learn two fundamental functional operations: map and reduce.
- ✯ **Relevant videos**:
  - Lambda function
  - Type inference
  - Metaprogramming

Information

1. Starting code: `https://github.com/ptal/functional`

Exercise 1 – Map operation

Implement a map function taking a vector-like type and applying a function `f` to each of its component. Suppose an integer vector `v` with the values `1,-5,6`, then `map(v, [](int x) { return x * 2; })` modifies `v` in-place and double each value. Use templates for the type of the vector and function.

Exercise 2 – Fold operation

Implement a left-fold function (aka. "reduce") which takes a vector, an accumulator and a function `f`. For instance, `fold_left(v, 0, [](int accu, int x) { return accu - x; })` returns the value `-2` (the difference of the integers in vector `v` from left to right). More information. Use templates for the types of the vector and function.

Exercise 3 – Fold right

Same as (2) but with `fold_right`, so it is from right to left. On the previous example, it also returns `-2`. Use templates for the type of the vector and function.

Exercise 4 – Rule 110 again

Implement the simulation loop of Rule 110 using map—think of the best way to get the index of the array instead of its elements. In addition, compute the size of the longest sequence of consecutive `1` occurring at any iteration, and print it. Implement this new feature using a fold operation.

Exercise 5 – Metaprogramming

Let us (partially) reimplement the type std::tuple. You only need to implement a constructor, destructor and the `get<i>(mytupl` free function. To implement `get`, you can use a helper member function:

```
template <class T, class... Ts>
struct Tuple {
  // ...
  template <int i>
  auto& get() {
    ...
  }
}

int main() {
  Tuple<double, int, int> t;
  // ...
  std::cout << get<0>(t) << std::endl;
  std::cout << get<1>(t) << std::endl;
  std::cout << get<2>(t) << std::endl;
}
```

(You have to define the helper structure `tuple_element`). Note that to call a template method, there is a special syntax. Unfortunately `tuple.get<0>()` will not work, you have to use `tuple.template get<0>()`, this is because the former syntax generates a parsing ambiguity.