

## Week 4

### Exercise 25

../25/insertable/insertable.h

```
1  #ifndef INCLUDED_INSERTABLE_
2  #define INCLUDED_INSERTABLE_
3
4  #define HDR_    template <typename Data, \
5                  template <typename, typename> class Container, \
6                  template <typename> class AllocationPolicy>
7  #define CONT_  Container<Data, AllocationPolicy<Data>>
8  #define INS_   Insertable<Data, Container, AllocationPolicy>
9  // Too many #defines?
10
11 #include <vector>
12 #include <memory>
13 #include <iterator>
14
15 template <typename Data,
16 template <typename, typename> class Container = std::vector,
17 template <typename> class AllocationPolicy = std::allocator>
18 class Insertable: public Container<Data, AllocationPolicy<Data>>
19 {
20     public:
21
22     Insertable();
23     Insertable(const CONT_ &RHS);
24     Insertable(const Insertable &RHS);
25     Insertable(Data RHS);
26
27     friend std::ostream &operator<< (std::ostream &out, const Insertable &ins)
28     {
29         std::copy (ins.begin(), ins.end(), std::ostream_iterator<Data>(out, "\n"));
30         return out;
31     };
32     // Don't yet know how to implment this without friend decl.
33 };
34
35
36 // Constructors just call constructor of underlying type
37 HDR_
38 INS_::Insertable()
39 : CONT_()
40 {};
41 HDR_
42 INS_::Insertable(const CONT_ &RHS)
43 : CONT_(RHS)
44 {};
45 HDR_
46 INS_::Insertable(const Insertable &RHS)
47 : CONT_(RHS)
48 {};
49 HDR_
50 INS_::Insertable(Data RHS)
51 : CONT_(RHS)
52 {};
53
54 #undef HDR_
55 #undef CONT_
56 #undef INS_
57 #endif
```

../25/main.ih

```
1 #define ERR(msg) printf("%s : %d", (msg), __LINE__)
2
3 using namespace std;
4
5 #include <iostream>
6
7 #include "insertable/insertable.h"
```

../25/main.cc

```
1 #include "main.ih"
2
3 int main(int argc, char const **argv)
4 {
5     typedef Insertable<int, std::vector> InsertableVector;
6     std::vector<int> vi {1, 2, 3, 4, 5};
7
8     InsertableVector iv;
9     InsertableVector iv2(vi);
10    InsertableVector iv3(4);
11    InsertableVector iv4(iv2);
12
13    cout << iv2 << '\n' <<
14          iv3 << '\n' <<
15          iv4 << '\n';
16
17    iv3.push_back(123);
18    cout << iv3 << '\n';
19
20 }
```

## Exercise 29

../29/expr.h

```
1  #ifndef INCLUDED_EXPRT_
2  #define INCLUDED_EXPRT_
3
4  #define EXPR_ template<typename LHS, \
5                  typename RHS, \
6                  template<typename> class Operation>
7
8  #include <cstdint>
9  #include <functional>
10
11  EXPR_
12  struct Expr;
13
14  template<typename RHS>
15  struct BasicType
16  {
17      typedef RHS ObjType;
18  };
19
20  EXPR_
21  struct BasicType<Expr<LHS, RHS, Operation>>
22  {
23      typedef typename Expr<LHS, RHS, Operation>::ObjType ObjType;
24  };
25
26  EXPR_
27  struct Expr
28  {
29      typedef typename BasicType<RHS>::ObjType ObjType;
30      typedef typename ObjType::value_type value_type;
31
32      LHS const &d_lhs;
33      RHS const &d_rhs;
34
35      Expr(LHS const &lhs, RHS const &rhs);
36
37      value_type operator[](size_t ix) const
38      {
39          static Operation<value_type> operation;
40          return operation(d_lhs[ix], d_rhs[ix]);
41      }
42
43      operator ObjType() const
44      {
45          ObjType retVal;
46          for (size_t ix = 0; ix != d_lhs.size(); ++ix)
47              retVal.push_back((*this)[ix]);
48          return retVal;
49      }
50  };
51
52  EXPR_
53  Expr<LHS, RHS, Operation>::Expr(LHS const &lhs, RHS const &rhs)
54  :
55      d_lhs(lhs),
56      d_rhs(rhs)
57  {};
58
59  #include "plusdeluxe.h"
60  template<typename LHS, typename RHS>
61  Expr<LHS, RHS, plusdeluxe> operator+(LHS const &lhs, RHS const &rhs)
62  {
```

```
63     return Expr<LHS, RHS, plusdeluxe>(lhs, rhs);
64 }
65
66 #undef EXPR_
67 #endif
```

../29/plusdeluxe.h

```
1  #ifndef INCLUDED_PLUSDELUXET_
2  #define INCLUDED_PLUSDELUXET_
3
4  template<typename RetType>
5  struct plusdeluxe
6  {
7      RetType operator()(const RetType &lhs, const RetType &rhs) const
8      {
9          return lhs + rhs;
10     }
11 };
12
13 #endif
```

## Exercise 30

../30/expr.h

```
1  #ifndef INCLUDED_EXPRT_
2  #define INCLUDED_EXPRT_
3
4  #define EXPR_ template<typename LHS, \
5                      typename RHS, \
6                      template<typename> class Operation>
7
8  #include <cstdint>
9  #include <functional>
10
11  EXPR_
12  struct Expr;
13
14  template<typename RHS>
15  struct BasicType
16  {
17      typedef RHS ObjType;
18  };
19
20  EXPR_
21  struct BasicType<Expr<LHS, RHS, Operation>>
22  {
23      typedef typename Expr<LHS, RHS, Operation>::ObjType ObjType;
24  };
25
26  EXPR_
27  struct Expr
28  {
29      typedef typename BasicType<RHS>::ObjType ObjType;
30      typedef typename ObjType::value_type value_type;
31
32      LHS const &d_lhs;
33      RHS const &d_rhs;
34
35      Expr(LHS const &lhs, RHS const &rhs);
36      size_t size() const;
37
38      value_type operator[](size_t ix) const
39      {
40          static Operation<value_type> operation;
41          return operation(d_lhs[ix], d_rhs[ix]);
42      }
43
44      operator ObjType() const
45      {
46          ObjType retVal;
47          for (size_t ix = 0; ix != d_lhs.size(); ++ix)
48              retVal.push_back((*this)[ix]);
49          return retVal;
50      }
51  };
52
53  EXPR_
54  size_t Expr<LHS, RHS, Operation>::size() const
55  {
56      return d_lhs.size();
57  };
58
59  EXPR_
60  Expr<LHS, RHS, Operation>::Expr(LHS const &lhs, RHS const &rhs)
61  :
62      d_lhs(lhs),
```

```
63     d_rhs(rhs)
64 };
65
66 template<typename LHS, typename RHS>
67 Expr<LHS, RHS, std::multiplies> operator*(LHS const &lhs, RHS const &rhs)
68 {
69     return Expr<LHS, RHS, std::multiplies>(lhs, rhs);
70 }
71
72 template<typename LHS, typename RHS>
73 Expr<LHS, RHS, std::plus> operator+(LHS const &lhs, RHS const &rhs)
74 {
75     return Expr<LHS, RHS, std::plus>(lhs, rhs);
76 }
77
78 template<typename LHS, typename RHS>
79 Expr<LHS, RHS, std::divides> operator/(LHS const &lhs, RHS const &rhs)
80 {
81     return Expr<LHS, RHS, std::divides>(lhs, rhs);
82 }
83
84 #undef Expr_
85 #endif
```

../30/main.ih

```
1  #define ERR(msg) printf("%s : %d", (msg), __LINE__)
2
3  #include "expr.h"
4  #include "printvector.h"
5
6  #include <vector>
7
8  template <typename T>
9  void print(T inputVector);
10
11 using namespace std;
```

../30/main.cc

```
1  #include "main.ih"
2
3  #include <vector>
4
5  int main()
6  {
7      using IVect = vector<int>;
8      IVect iv1(10, 4);           // IVect: vector<int>
9      IVect iv2(10, 3);
10     IVect iv3(10, 2);
11     IVect iv4(10, 1);
12
13     IVect iResult { iv1 * (iv2 + iv3) / iv4 };
14
15     using DVect = vector<double>;
16     DVect dv1(10, 4.1);         // DVect: vector<double>
17     DVect dv2(10, 3.1);
18     DVect dv3(10, 2.1);
19     DVect dv4(10, 1.1);
20
21     DVect dResult { dv1 * (dv2 + dv3) / dv4 };
22
23     print(dv1);
24     print(dResult);
25 }
```