Week 4

Exercise 25

../25/insertable/insertable.h

```
#ifndef INCLUDED_INSERTABLE_
#define INCLUDED_INSERTABLE_

#define HDR_  template <typename Data, \
              template <typename, typename> class Container, \
              template <typename> class AllocationPolicy>
#define CONT_ Container<Data, AllocationPolicy<Data>>
#define INS_  Insertable<Data, Container, AllocationPolicy>
// Too many #defines?

#include <vector>
#include <memory>
#include <iterator>

template <typename Data,
template <typename, typename> class Container = std::vector,
template <typename> class AllocationPolicy = std::allocator>
class Insertable: public Container<Data, AllocationPolicy<Data>>
{
  public:

    Insertable();
    Insertable(const CONT_ &RHS);
    Insertable(const Insertable &RHS);
    Insertable(Data RHS);

    friend std::ostream &operator<<(std::ostream &out, const Insertable &ins)
    {
      std::copy (ins.begin(), ins.end(), std::ostream_iterator<Data>(out, "\n"));
      return out;
    };
    // Don't yet know how to implmenent this without friend decl.
};


// Constructors just call constructor of underlying type
HDR_
INS_::Insertable()
: CONT_()
{};
HDR_
INS_::Insertable(const CONT_ &RHS)
  : CONT_(RHS)
{};
HDR_
INS_::Insertable(const Insertable &RHS)
  : CONT_(RHS)
{};
HDR_
INS_::Insertable(Data RHS)
  : CONT_(RHS)
{};

#undef HDR_
#undef CONT_
#undef INS_
#endif
```

../25/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

using namespace std;

#include <iostream>

#include "insertable/insertable.h"
```

../25/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  typedef Insertable<int, std::vector> InsertableVector;
  std::vector<int> vi {1, 2, 3, 4, 5};

  InsertableVector iv;
  InsertableVector iv2(vi);
  InsertableVector iv3(4);
  InsertableVector iv4(iv2);

  cout << iv2 << '\n' <<
          iv3 << '\n' <<
          iv4 << '\n';

  iv3.push_back(123);
  cout << iv3 << '\n';

}
```

Exercise 29

../29/expr.h

```
#ifndef INCLUDED_EXPRT_
#define INCLUDED_EXPRT_

#define EXPR_ template<typename LHS, \
                  typename RHS, \
                  template<typename> class Operation>

#include <cstddef>
#include <functional>

EXPR_
struct Expr;

template<typename RHS>
struct BasicType
{
  typedef RHS ObjType;
};

EXPR_
struct BasicType<Expr<LHS, RHS, Operation>>
{
  typedef typename Expr<LHS, RHS, Operation>::ObjType ObjType;
};

EXPR_
struct Expr
{
  typedef typename BasicType<RHS>::ObjType ObjType;
  typedef typename ObjType::value_type value_type;

  LHS const &d_lhs;
  RHS const &d_rhs;

  Expr(LHS const &lhs, RHS const &rhs);

  value_type operator[](size_t ix) const
  {
    static Operation<value_type> operation;
    return operation(d_lhs[ix], d_rhs[ix]);
  }

  operator ObjType() const
  {
    ObjType retVal;
    for (size_t ix = 0; ix != d_lhs.size(); ++ix)
      retVal.push_back((*this)[ix]);
    return retVal;
  }
};

EXPR_
Expr<LHS, RHS, Operation>::Expr(LHS const &lhs, RHS const &rhs)
:
  d_lhs(lhs),
  d_rhs(rhs)
{};

#include "plusdeluxe.h"
template<typename LHS, typename RHS>
Expr<LHS, RHS, plusdeluxe> operator+(LHS const &lhs, RHS const &rhs)
{
  return Expr<LHS, RHS, plusdeluxe>(lhs, rhs);
}

#undef EXPR_
#endif
```

../29/plusdeluxe.h

```
#ifndef INCLUDED_PLUSDELUXET_
#define INCLUDED_PLUSDELUXET_

template<typename RetType>
struct plusdeluxe
{
  RetType operator()(const RetType &lhs, const RetType &rhs) const
  {
    return lhs + rhs;
  }
};

#endif
```

Exercise 30

../30/expr.h

```
#ifndef INCLUDED_EXPRT_
#define INCLUDED_EXPRT_

#define EXPR_ template<typename LHS, \
                       typename RHS, \
                       template<typename> class Operation>

#include <cstddef>
#include <functional>

EXPR_
struct Expr;

template<typename RHS>
struct BasicType
{
  typedef RHS ObjType;
};

EXPR_
struct BasicType<Expr<LHS, RHS, Operation>>
{
  typedef typename Expr<LHS, RHS, Operation>::ObjType ObjType;
};

EXPR_
struct Expr
{
  typedef typename BasicType<RHS>::ObjType ObjType;
  typedef typename ObjType::value_type value_type;

  LHS const &d_lhs;
  RHS const &d_rhs;

  Expr(LHS const &lhs, RHS const &rhs);
  size_t size() const;

  value_type operator[](size_t ix) const
  {
    static Operation<value_type> operation;
    return operation(d_lhs[ix], d_rhs[ix]);
  }

  operator ObjType() const
  {
    ObjType retVal;
    for (size_t ix = 0; ix != d_lhs.size(); ++ix)
      retVal.push_back((*this)[ix]);
    return retVal;
  }
};

EXPR_
size_t Expr<LHS, RHS, Operation>::size() const
{
  return d_lhs.size();
};

EXPR_
Expr<LHS, RHS, Operation>::Expr(LHS const &lhs, RHS const &rhs)
:
  d_lhs(lhs),
  d_rhs(rhs)
{};

template<typename LHS, typename RHS>
Expr<LHS, RHS, std::multiplies> operator*(LHS const &lhs, RHS const &rhs)
{
  return Expr<LHS, RHS, std::multiplies>(lhs, rhs);
}

template<typename LHS, typename RHS>
Expr<LHS, RHS, std::plus> operator+(LHS const &lhs, RHS const &rhs)
{
  return Expr<LHS, RHS, std::plus>(lhs, rhs);
}

template<typename LHS, typename RHS>
Expr<LHS, RHS, std::divides> operator/(LHS const &lhs, RHS const &rhs)
{
  return Expr<LHS, RHS, std::divides>(lhs, rhs);
}

#undef EXPR_
#endif
```

../30/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

#include "expr.h"
#include "printvector.h"

#include <vector>

template <typename T>
void print(T inputVector);

using namespace std;
```

../30/main.cc

```
#include "main.ih"

#include <vector>

int main()
{
  using IVect = vector<int>;
  IVect iv1(10, 4);        // IVect: vector<int>
  IVect iv2(10, 3);
  IVect iv3(10, 2);
  IVect iv4(10, 1);

  IVect iResult { iv1 * (iv2 + iv3) / iv4 };

  using DVect = vector<double>;
  DVect dv1(10, 4.1);     // DVect: vector<double
  DVect dv2(10, 3.1);
  DVect dv3(10, 2.1);
  DVect dv4(10, 1.1);

  DVect dResult { dv1 * (dv2 + dv3) / dv4 };

  print(dv1);
  print(dResult);
}
```