Programming in C/C++
Exercises set one: class templates

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

January 27, 2017

Exercise 1, new matrix

In this exercise we changed the matrix class to work with templates. Keep in mind that this would make the header file humongous as the entirety of the template class needs to be in the header. Since we had all the functions in seperate files anyways we just kind of left them there and only included the destructor. Just imagine all the functions as being place in the header file as the destructor is with no seperate declaration and definition.

Code listings

Listing 1: matrix.ih

```
#include "matrix.h"

#include <cstring>
#include <iostream>

using namespace std;
```

Listing 2: matrix.h

```
#ifndef INCLUDED_MATRIX_
#define INCLUDED_MATRIX_

#include <iosfwd>
#include <initializer_list>
```

```

template <typename Type>
class Matrix
{
    size_t d_nRows = 0;
    size_t d_nCols = 0;
    Type *d_data = 0;

    class Proxy
    {
        friend class Matrix;
  template <typename U>
        friend std::istream &operator>>(std::istream &in, Proxy &&prox);

        Matrix &d_mat;

        int d_direction = Matrix::BY_ROWS;
        size_t d_from = 0;
        size_t d_count = ~0UL;
        size_t d_nRows;
        size_t d_nCols;

        Proxy(Matrix &mat, int extractionType, size_t from,
                size_t count, size_t nRows, size_t nCols);

        std::istream &extractFrom(std::istream &in);
        std::istream &extractRows(std::istream &in);
        std::istream &extractCols(std::istream &in);
    };

    friend class Proxy;

    template <typename U>
    friend std::istream &operator>>(std::istream &in, Proxy &&mat);


    public:
        enum Extraction
        {
            BY_ROWS,
            BY_COLS
        };

        typedef std::initializer_list
            <std::initializer_list<Type>> IniList;

        Matrix() = default;
        Matrix(size_t nRows, size_t nCols);
        Matrix(Matrix const &other);
        Matrix(Matrix &&tmp);
        Matrix(IniList inilist);

        ~Matrix()
        {
      delete d_data;
    }

        Matrix &operator=(Matrix const &rhs);
        Matrix &operator=(Matrix &&tmp);


        size_t nRows() const;
        size_t nCols() const;
        size_t size() const;

        static Matrix identity(size_t dim);

        Matrix &tr();
        Matrix transpose() const;

        void swap(Matrix &other);

        Type *operator[](size_t idx);
        Type const *operator[](size_t idx) const;

            Matrix &operator+=(Matrix const &rhs)   &;
            Matrix &&operator+=(Matrix const &rhs) &&;

        Proxy operator()(size_t nRows, size_t nCols,
                         Extraction type = BY_ROWS);
        Proxy operator()(Extraction type, size_t
            from = 0, size_t count = ~0UL);

    private:
        template <typename U>
        friend Matrix operator+(Matrix const &lhs,
   Matrix const &rhs);

    template <typename U>
        friend Matrix operator+(Matrix &&lhs, Matrix
   const &rhs);

        void add(Matrix const &rhs);

        size_t extractionLimits(size_t from, size_t
   count, size_t available);
        void setDimensions(size_t nRows,
            size_t nCols);

        template <typename U>
        friend bool operator==(Matrix const &lhs,
   Matrix const &rhs);

        Type &el(size_t row, size_t col) const;
};

template <typename Type>
std::ostream &operator<<(std::ostream &out,
    Matrix<Type> const &mat);
template <typename Type>
std::istream &operator>>(std::istream &in,
    Matrix<Type> &mat);

template <typename Type>
inline bool operator!=(Matrix<Type> const &lhs,
    Matrix<Type> const &rhs)
```

```cpp
{
    return not (lhs == rhs);
}

template <typename Type>
inline Type *Matrix<Type>::operator[](size_t idx)
{
    return &el(idx, 0);
}

template <typename Type>
inline Type const *Matrix<Type>::operator[](size_t idx
   ) const
{
    return &el(idx, 0);
}
template <typename Type>
inline size_t Matrix<Type>::nCols() const
{
    return d_nCols;
}
template <typename Type>
inline size_t Matrix<Type>::nRows() const
{
    return d_nRows;
}
template <typename Type>
inline size_t Matrix<Type>::size() const
{
    return d_nRows * d_nCols;
}

template <typename Type>
inline Type &Matrix<Type>::el(size_t row, size_t col)
   const
{
    return d_data[row * d_nCols + col];
}

#endif
```

Listing 3: main.cc

```
#include "matrix.ih"

int main()
{
    Matrix<int> mx;
}
```

matrix files

Listing 4: matrix1.cc

```
#include "matrix.ih"

template <typename Type>
Matrix<Type>::Matrix(size_t nRows, size_t nCols)
:
    d_nRows(nRows),
    d_nCols(nCols),
    d_data(new Type[size()]())
{}
```

Listing 5: matrix2.cc

```
#include "matrix.ih"

template <typename Type>
Matrix<Type>::Matrix(Matrix const &other)
:
    d_nRows(other.d_nRows),
    d_nCols(other.d_nCols),
    d_data(new Type[size()])
{
    memcpy(d_data, other.d_data, size() *
    sizeof(Type));
}
```

Listing 6: matrix3.cc

```
#include "matrix.ih"

template <typename Type>
```

```
Matrix<Type>::Matrix(Matrix &&tmp)
{
    swap(tmp);
}
```

Listing 7: matrix4.cc

```
#include "matrix.ih"

template <typename Type>
Matrix<Type>::Matrix(IniList iniList)
:
    d_nRows(iniList.size()),
    d_nCols(iniList.begin()->size()),
    d_data(new Type[size()])
{
    auto ptr = d_data;
    for (auto &list: iniList)
    {
        if (list.size() != d_nCols)
        {
            cerr << "Matrix(IniList): varying"
            " number of elements in rows\n";
            exit(1);
        }
        memcpy(ptr, &*list.begin() , list.size() *
   sizeof(Type));
        ptr += list.size();
    }
}
```

Listing 8: add.cc

```
#include "matrix.ih"

template <typename Type>
void Matrix<Type>::add(Matrix const &rhs)
{
    if (d_nRows != rhs.d_nRows || d_nCols
        != rhs.d_nCols)
    {
        cerr << "Cannot add matrices"
        " of unequal dimensions\n";
        exit(1);
    }

    for (size_t idx = 0, end = size(); idx != end;
    ++idx)
        d_data[idx] += rhs.d_data[idx];
}
```

Listing 9: operatoradd1.cc

```
#include "matrix.ih"

template <typename Type>
Matrix<Type> operator+(Matrix<Type> const &lhs, Matrix
   <Type> const &rhs)
{
    Matrix<Type> ret(lhs);
    ret.add(rhs);
    return ret;
}
```

Listing 10: operatoradd2.cc

```
#include "matrix.ih"

template <typename Type>
Matrix<Type> operator+(Matrix<Type> &&lhs,
    Matrix<Type> const &rhs)
{
    Matrix<Type> ret(move(lhs));
    ret.add(rhs);
    return ret;
}
```

Listing 11: operatoraddis1.cc

```
#include "matrix.ih"

template <typename Type>
Matrix<Type> &Matrix<Type>::operator+=(Matrix const &
   rhs) &
```

```
{
    add(rhs);
    return *this;
}
```

Listing 12: operatoraddis2.cc

```
#include "matrix.ih"

template <typename Type>
Matrix<Type> &&Matrix<Type>::operator+=(Matrix const &
    rhs) &&
{
    add(rhs);
    return move(*this);
}
```

Listing 13: operatorassign1.cc

```
#include "matrix.ih"

template <typename Type>
Matrix<Type> &Matrix<Type>::operator=(Matrix const &
    other)
{
    Matrix tmp(other);
    swap(tmp);
    return *this;
}
```

Listing 14: operatorassign2.cc

```
#include "matrix.ih"

template <typename Type>
Matrix<Type> &Matrix<Type>::operator=(Matrix &&tmp)
{
    swap(tmp);
    return *this;
}
```

Listing 15: operatorequal.cc

```cpp
#include "matrix.ih"

template <typename Type>
bool operator==(Matrix<Type> const &lhs, Matrix<Type>
   const &rhs)
{
    if (lhs.d_nRows != rhs.d_nRows || lhs.d_nCols !=
   rhs.d_nCols)
        return false;

    for (size_t idx = 0, end = lhs.size(); idx != end;
    ++idx)
    {
        if (lhs.d_data[idx] != rhs.d_data[idx])
            return false;
    }

    return true;
}
```

Listing 16: operatorfun1.cc

```cpp
#include "matrix.ih"

template <typename Type>
typename Matrix<Type>::Proxy Matrix<Type>::operator()(
   size_t nRows, size_t nCols, Extraction type)
{
    return type == BY_ROWS ?
        Proxy{*this, BY_ROWS, 0, nRows, nRows, nCols}
    :
        Proxy{*this, BY_COLS, 0, nCols, nRows, nCols};
}
```

Listing 17: operatorfun2.cc

```cpp
#include "matrix.ih"

template <typename Type>
typename Matrix<Type>::Proxy Matrix<Type>::operator()(
   Extraction type, size_t from, size_t count)
```

```
{
    return type == BY_ROWS ?
        Proxy{*this, BY_ROWS, from, extractionLimits(
   from, count, d_nRows),

   d_nRows, d_nCols}
     :
        Proxy{*this, BY_COLS, from, extractionLimits(
   from, count, d_nCols),

   d_nRows, d_nCols};
}
```

proxy files

Listing 18: proxy1.cc

```
#include "matrix.ih"

template <typename Type>
Matrix<Type>::Proxy::Proxy(Matrix &mat, int
   extractionType, size_t from,
                    size_t count, size_t nRows,
   size_t nCols)
:
    d_mat(mat),
    d_direction(extractionType),
    d_from(from),
    d_count(count),
    d_nRows(nRows),
    d_nCols(nCols)
{}
```

Listing 19: proxyextractcols.cc

```
#include "matrix.ih"

template <typename Type>
istream &Matrix<Type>::Proxy::extractCols(istream &in)
{
    d_mat.setDimensions(d_nRows, d_nCols);

    for (; d_count--; ++d_from)
    {
        for (size_t row = 0, end = d_mat.nRows(); row
   != end; ++row)
            in >> d_mat[row][d_from];
    }

    return in;
}
```

Listing 20: proxyextractfrom.cc

```
#include "matrix.ih"

template <typename Type>
istream &Matrix<Type>::Proxy::extractFrom(istream &in)
{
    return d_direction == Matrix<Type>::BY_ROWS ?
   extractRows(in) : extractCols(in);
}
```

Listing 21: proxyextractrows.cc

```
#include "matrix.ih"

template <typename Type>
istream &Matrix<Type>::Proxy::extractRows(istream &in)
{
    d_mat.setDimensions(d_nRows, d_nCols);

    for (; d_count--; ++d_from)
    {
        auto rowPtr = d_mat[d_from];
        for (size_t col = 0, end = d_mat.nCols(); col
   != end; ++col)
            in >> rowPtr[col];
    }
    return in;
}
```

Exercise 2, Member template

In this exercise we changed one of the methods of the Semaphore class to make it a member template using perfect forwarding.

Code listings

Listing 22: semaphore.ih

```
#include "semaphore.h"
using namespace std;
```

Listing 23: semaphore.h

```
#ifndef INCLUDED_SEMAPHORE_
#define INCLUDED_SEMAPHORE_

#include <functional>
#include <mutex>
#include <condition_variable>

class Semaphore
{
    mutable std::mutex d_mutex;
    std::condition_variable d_condition;
    size_t d_nAvailable;

 public:
    Semaphore(size_t nAvailable);

    template <typename Function, typename ...Params>
  bool wait(Function fun, Params &&...params)
    {
        fun(std::forward<Params>(params)...);
        std::unique_lock<std::mutex> lk(d_mutex);

        while (d_nAvailable == 0)
            d_condition.wait(lk);

        if (d_nAvailable == 1 &&
            not fun(d_nAvailable))
```

```
                return false;

            --d_nAvailable;

            return true;
        };

        void notify_all();
        size_t size() const;
};

#endif
```

Listing 24: constructor.cc

```
#include "semaphore.ih"

Semaphore::Semaphore(size_t nAvailable)
    :
    d_nAvailable(nAvailable)
{}
```

Listing 25: notify.cc

```
#include "semaphore.ih"

void Semaphore::notify_all()
{
    lock_guard<mutex> lk(d_mutex);
    if (d_nAvailable++ == 0)
        d_condition.notify_all();
}
```

Listing 26: size.cc

```
#include "semaphore.ih"

size_t Semaphore::size() const
{
    return d_nAvailable;
}
```

Exercise 3, custom back inserter

In this exercise we make a custom class work with the `back_inserter` iterator so we can use the `copy` generic algorithm.

Code listings

Listing 27: data.ih

```
#include "data.h"
#include <algorithm>
#include <iterator>

using namespace std;
```

Listing 28: data.h

```
#ifndef DATA_H
#define DTA_H

#include <vector>
#include <memory>
#include <iostream>

class Data
{
  typedef std::vector<std::shared_ptr<
    std::string>> DataVector;

  DataVector d_data;

  public:
    typedef std::string value_type;
    void push_back(std::string const &str);
    void vecOutput();
};

#endif
```

Listing 29: main.cc

```
#include "data.ih"
```

```

int main(int argc, char **argv)
{
  Data DataObj;
  copy(istream_iterator<string>(cin),
    istream_iterator<string>(),
    back_inserter(DataObj));
  DataObj.vecOutput();
}
```

Listing 30: pushback.cc

```
#include "data.ih"

void Data::push_back(string const &str)
{
  shared_ptr<string> somePtr =
    make_shared<string>(str);

  d_data.push_back(somePtr);
}
```

Exercise 5, static polymorphism

We made a static polymorphic class that prints things!

Code listings

Listing 31: inserter.ih

```
#include "inserter.h"

using namespace std;
```

Listing 32: inserter.h

```
#ifndef INSERTER_H
#define INSERTER_H

#include <iostream>

template <typename Derived>
```

```
class Inserter
{
  private:
    std::ostream &insertInto(std::ostream &out)
    {
      return static_cast<Derived*>(this)->
        insertInto(out);
    }

  template <typename Derivative>
  friend std::ostream &operator<<(std::ostream &out,
    Inserter<Derivative> &base);
};

template <typename Derivative>
std::ostream &operator<<(std::ostream &out,
  Inserter<Derivative> &base)
{
  return base.insertInto(out);
}

#endif
```

Listing 33: main.ih

```
#include "main.h"

using namespace std;
```

Listing 34: main.h

```
#ifndef MAIN_H
#define MAIN_H

#include "inserter.h"

class IntValue : public Inserter<IntValue>
{
  int d_int;

  public:
    IntValue(int someInt);

  private:
    std::ostream &insertInto(std::ostream &out);

  friend Inserter;
};

class DoubleValue : public Inserter<DoubleValue>
{
  double d_double;

  public:
    DoubleValue(double someDouble);

  private:
    std::ostream &insertInto(std::ostream &out);

  friend Inserter;
};

#endif
```

Listing 35: main.cc

```
#include "main.ih"

int main(int argc, char **argv)
{
  IntValue iv(12);
  DoubleValue dv(3.14);

  cout << iv << '\n';
  cout << dv << '\n';
}
```

IntValue

Listing 36: intconstructor.cc

```
#include "main.ih"

```

```
IntValue::IntValue(int someInt)
:
  d_int(someInt)
{
}
```

Listing 37: intinserter.cc

```
#include "main.ih"

ostream &IntValue::insertInto(ostream &out)
{
  return out << d_int;
}
```

DoubleValue

Listing 38: doubleconstructor.cc

```
#include "main.ih"

DoubleValue::DoubleValue(double someDouble)
:
  d_double(someDouble)
{
}
```

Listing 39: doubleinserter.cc

```
#include "main.ih"

ostream &DoubleValue::insertInto(ostream &out)
{
  return out << d_double;
}
```

Exercise 6, static polymorphism contd.

Now with more inheritence?

Code listings

Listing 40: main.ih

```cpp
#include "main.h"

using namespace std;
```

Listing 41: main.h

```cpp
#ifndef MAIN_H
#define MAIN_H

#include "inserter.h"

class IntValue : public Inserter<IntValue>
{
  int d_int;

  public:
    IntValue(int someInt);
    int value();

  private:
    virtual std::ostream &insertInto(
      std::ostream &out);

  friend Inserter;
};

class DoubleValue : public Inserter<DoubleValue>
{
  double d_double;

  public:
    DoubleValue(double someDouble);

  private:
    std::ostream &insertInto(std::ostream &out);

  friend Inserter;
};

```

```
class LabelledInt : public IntValue
{
  std::string d_label;

  public:
    LabelledInt(int someInt, std::string label);

  private:
    std::ostream &insertInto(
      std::ostream &out) override;

  friend Inserter;
};

#endif
```

Listing 42: main.cc

```
#include "main.ih"

int main(int argc, char **argv)
{
  IntValue iv(12);
  DoubleValue dv(3.14);
  LabelledInt li(3, "lithium");

  cout << iv << '\n';
  cout << dv << '\n';
  cout << li << '\n';
}
```

LabelledInt

Listing 43: labelconstructor.cc

```
#include "main.ih"

LabelledInt::LabelledInt(int someInt, string label)
:
  IntValue(someInt),
  d_label(label)
{
}
```

Listing 44: labelinserter.cc

```
#include "main.ih"

ostream &LabelledInt::insertInto(ostream &out)
{
  return out << d_label << ": " << value();
}
```