

Programming in C/C++

Exercises set two: inheritance

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

November 23, 2016

Exercise 9, addition/subtraction hierarchy header files

These are the final header files for this class hierarchy.

Design problem

The prototypical design problem here is that we want use private members of another class. The solution to this is to make `Operations` friends with `Addition` and `Subtraction`. This is the solution because there is no other way to carry over friendship like this, friendship is not transitive and not carried over by inheritance.

Code listings

Listing 1: addition.ih

```
1 #include "addition.h"
2 #include <iostream>
3
4 using namespace std;
```

Listing 2: addition.h

```
1 #ifndef ADDITION_H
2 #define ADDITION_H
3
4 class Operations;
```

```

5
6 class Addition
7 {
8     public:
9         Addition &operator+=(Operations const &rhs);
10 };
11
12 Operations operator+(Operations const &lhs,
13     Operations const &rhs);
14
15 #endif

```

Listing 3: binops.ih

```

1 #include "binops.h"

```

Listing 4: binops.h

```

1 #ifndef BINOPS_H
2 #define BINOPS_H
3
4 #include "../addition/addition.h"
5 #include "../subtraction/subtraction.h"
6
7 class Binops: public Addition, public Subtraction
8 {
9     public:
10         ~Binops() {};
11 };
12
13 #endif

```

Listing 5: subtraction.ih

```

1 #include "subtraction.h"
2 #include <iostream>
3
4 using namespace std;

```

Listing 6: subtraction.h

```

1 #ifndef SUBTRACTION_H
2 #define SUBTRACTION_H

```

```

3
4 class Operations;
5
6 class Subtraction
7 {
8     public:
9         Subtraction &operator==(
10             Operations const &rhs);
11 };
12
13 Operations operator-(Operations const &lhs,
14     Operations const &rhs);
15
16 #endif

```

Listing 7: operations.ih

```

1 #include "operations.h"
2 #include <iostream>
3
4 using namespace std;

```

Listing 8: operations.h

```

1 #ifndef OPERATIONS_H_
2 #define OPERATIONS_H_
3
4 #include "../binops/binops.h"
5
6 class Operations: public Binops
7 {
8     friend Binops;
9     friend Addition;
10    friend Subtraction;
11
12    friend Operations operator+(Operations const &lhs,
13        Operations const &rhs);
14    friend Operations operator-(Operations const &lhs,
15        Operations const &rhs);
16
17    void add(Operations const &rhs);

```

```

18     void sub(Operations const &rhs);
19
20     public:
21         Operations() = default;
22         Operations(Addition const &other);
23         Operations(Subtraction const &other);
24 };
25
26 #endif

```

Exercise 10, compound assignment operators

Here we implement the compound assignment operators in the class hierarchy.

Code listings

Listing 9: operator+=.cc

```

1 #include "addition.ih"
2 #include "../operations/operations.h"
3
4 Addition &Addition::operator+=(Operations const &rhs)
5 {
6     cout << "compound addition\n";
7     Operations copy(*this);
8     copy.add(rhs);
9     return *this;
10 }

```

Listing 10: operator-=.cc

```

1 #include "subtraction.ih"
2 #include "../operations/operations.h"
3
4 Subtraction &Subtraction::operator-=(
5     Operations const &rhs)
6 {
7     cout << "compound subtraction\n";
8     Operations copy(*this);
9     copy.sub(rhs);
10    return *this;
11 }

```

Exercise 11, free binary operators

Here we implement the free binary operators in the class hierarchy as well as the add and sub members.

Code listings

Listing 11: main.cc

```
1  #include "main.h"
2
3  int main(int argc, char **argv)
4  {
5      Operations op1;
6      Operations op2;
7
8      op1 += op2;
9      op2 -= op1;
10
11     Operations op3 = op1 + op2;
12     Operations op4 = op2 - op1;
13 }
```

Listing 12: operator+.cc

```
1  #include "../operations/operations.h"
2  #include <iostream>
3
4  using namespace std;
5
6  Operations operator+(Operations const &lhs,
7      Operations const &rhs)
8  {
9      cout << "binary addition\n";
10     Operations copy(lhs);
11     copy.add(rhs);
12     return copy;
13 }
```

Listing 13: operator-.cc

```
1  #include "../operations/operations.h"
```

```

2  #include <iostream>
3
4  using namespace std;
5
6  Operations operator-(Operations const &lhs,
7    Operations const &rhs)
8  {
9    cout << "binary subtraction\n";
10   Operations copy(lhs);
11   copy.sub(rhs);
12   return copy;
13 }

```

Listing 14: add.cc

```

1  #include "operations.ih"
2
3  void Operations::add(Operations const &rhs)
4  {
5    cout << "addition\n";
6  }

```

Listing 15: sub.cc

```

1  #include "operations.ih"
2
3  void Operations::sub(Operations const &rhs)
4  {
5    cout << "subtraction\n";
6  }

```

Exercise 12, matrix copying

We were tasked with initializing an array with copies of a matrix `mat` with only a new statement.

Solution

Because the default constructor is always called we made a new class that is derived from `matrix`. This `Copymatrix` holds a static `Matrix` that is used to initialize the class. By returning a `Matrix` pointer the unnecessary data is sliced off.

Code listings

Listing 16: main.ih

```
1 #include "main.h"
2
3 using namespace std;
```

Listing 17: main.h

```
1 #ifndef MAIN_H_
2 #define MAIN_H_
3
4 #include "matrixmaker.h"
5
6 #endif
```

Listing 18: factory.cc

```
1 #include "matrixmaker.h"
2
3 namespace MatrixMaker
4 {
5     Matrix const *MatrixMaker::CopyMatrix::
6         d_blueprint = 0;
7
8     Matrix *factory(Matrix const &mat, size_t count)
9     {
10         MatrixMaker::CopyMatrix::d_blueprint = &mat;
11         return new MatrixMaker::CopyMatrix[count];
12     }
13 }
```

Listing 19: main.cc

```
1 #include "main.ih"
2
3 int main(int argc, char **argv)
4 {
5     Matrix mat({{0, 1, 2, 3,}, {4, 5, 6, 7}});
6
7     size_t count = 8;
8     Matrix *matArray = MatrixMaker::factory(mat, count);
```

```

9
10     for (size_t index = 0; index != count; ++index)
11     {
12         cout << matArray[index]
13             << '\n';
14     }
15
16     delete[] matArray;
17 }

```

CopyMatrix

Listing 20: copymatrix.h

```

1  #ifndef MATRIXMAKER_H_
2  #define MATRIXMAKER_H_
3
4  #include "matrix/matrix.h"
5
6  namespace MatrixMaker
7  {
8      class CopyMatrix: public Matrix
9      {
10     public:
11         CopyMatrix();
12
13         static Matrix const *d_blueprint;
14     };
15
16     Matrix *factory(Matrix const &mat, size_t count);
17 }
18
19 #endif

```

Listing 21: copymatrix1.cc

```

1  #include "matrixmaker.h"
2
3  namespace MatrixMaker
4  {
5      CopyMatrix::CopyMatrix()
6      :

```



```

7     Matrix(*d_blueprint)
8     {
9     }
10 }

```

Exercise 13, red thread

We were tasked with using inheritance to include the `Limits` class into a number of other classes.

Code listings

Listing 22: fighter.h

```

1  #ifndef INCLUDED_FIGHTER_
2  #define INCLUDED_FIGHTER_
3
4  #include "../limits/limits.h"
5  #include "../time/time.h"
6  #include "../coordinates/coordinates.h"
7  #include "../speed/speed.h"
8  #include "../altitude/altitude.h"
9  #include "../heading/heading.h"
10 #include "../registerdata/registerdata.h"
11 #include "../units/units.h"
12
13
14 class Fighter: private Limits
15 {
16     RegisterData d_rd;
17
18     // keeps track of time-related info
19     Time         d_time;
20     Units         d_units;
21     Coordinates d_coord;
22     Speed         d_speed;
23     Altitude      d_altitude;
24     Heading       d_heading;
25
26     // true: inside the box
27     bool         d_inTheBox = false;

```

```

28
29     static size_t s_nFighters;
30     static size_t s_nRegisteredFighters;
31
32     public:
33         Fighter(RegisterData const &rd,
34             // 1
35             int xCoord, int yCoord, int units);
36         ~Fighter();
37
38         void setUnits(int type);
39
40         void altitudeTo(size_t altitude, size_t rate);
41         void headingTo(char direction, size_t hdg,
42             double acceleration);
43         void speedTo(size_t kts);
44
45         // default: no update time changes
46         void info(size_t silentTime= 0);
47     private:
48         void boxStatus();
49 };
50
51 #endif

```

Listing 23: fighter's altitudeto.cc

```

1  #include "fighter.i.h"
2
3  void Fighter::altitudeTo(size_t req, size_t rate)
4  {
5      d_altitude.set(
6          d_units.setAlt(req),
7          rate == 0 ? DEFAULT_CLIMBRATE
8          : d_units.setRate(rate)
9      );
10 }

```

Listing 24: monitor.h

```

1  #ifndef INCLUDED_MONITOR_

```

```

2 #define INCLUDED_MONITOR_
3
4     // messages to the monitor are received
5     // on fifo '0'
6
7 #include <string>
8
9 #include "../limits/limits.h"
10 #include "../fightermap/fightermap.h"
11
12 class Monitor: private Limits
13 {
14     FighterMap d_fighter;
15     std::string d_fifo;
16
17     public:
18         Monitor(char const *dir);
19         Monitor(Monitor const &other) = delete;
20
21         void run();
22
23     private:
24         // 1st char already removed
25         void insert(std::istreamstream &instr);
26         // 1st char already removed
27         void remove(std::istreamstream &instr);
28 };
29
30 #endif

```

Listing 25: time.h

```

1 #ifndef INCLUDED_TIME_
2 #define INCLUDED_TIME_
3
4 #include <iosfwd>
5
6 #include "../limits/limits.h"
7
8 class Time: private Limits
9 {

```

```

10     size_t d_TOtime = 0;    // take-off time
11
12     // time elapsed since the previous update;
13     size_t d_delta;
14     size_t d_time;
15     // time in seconds at the last update
16
17     // clock-time set by updateTime()
18     static size_t s_sec;
19
20     public:
21         Time();
22         // update the time for a Fighter
23         void step();
24         // since take-off
25         size_t elapsed() const;
26         void registerTOtime();
27         size_t delta() const;
28         size_t fuelRemaining() const;
29         // called by Monitor::childProcess before
30         // updating the Fighters' data
31         static void updateTime();
32
33         // returns the common clock-time
34         static size_t clock();
35 };
36
37 inline Time::Time()
38 :
39     d_time(s_sec)
40 {}
41
42 inline size_t Time::delta() const
43 {
44     return d_delta;
45 }
46
47 inline size_t Time::elapsed() const
48 {
49     return d_time - d_TOtime;

```

```
50 }
51
52 inline size_t Time::fuelRemaining() const
53 {
54     return (FUEL_EMPTY - elapsed()) / 60;
55 }
56
57 inline size_t Time::clock()
58 {
59     return s_sec;
60 }
61
62 #endif
```