

Exercises week 3

Klaas Isaac Bijlsma
s2394480

David Vroom
s2309939

December 6, 2017

Exercise 20

Learn the implications of using friends

The exercise states that the classes `Addition` and `Subtraction`, which implement the binary addition and subtraction operators, are base classes of a class `Binops`. Furthermore, a class `Operations` implements the private functions `add` and `sub`, and inherits from `Binops`. Also, `Operations` declares `Binops` a friend class.

- An operator like `Addition::operator+=` has to call the private `add` member of `Operations`, but `Operations` does not declare `Addition` as a friend class.
We solved this problem by defining a wrapper function `binopsAdd` in `Binops`, which calls `add`. This is allowed, since `Binops` is a friend class of `Operations`. Then, `Addition` is declared a friend class of `Binops`, which is therefore allowed to call `binopsAdd`, and by that it calls `add`. Ommiting the wrapper function does not work, since `Addition`, a friend class of a friend class of `Operations`, can not call `add`. The problem is handled in a similar way for `Subtraction`.
- There is however a fundamental problem with this design. By defining the class hierarchy as stated in the exercise, multiple friendships among classes have to be defined. In this way, a strong coupling between the classes is obtained, which is undesirable.
- Below we provide the class interfaces of `Binops`, `Addition` and `Subtraction`.

`binops/binops.h`

```
1 | #ifndef EX20_BINOPS_H
```

```

2  #define EX20_BINOPS_H
3
4  #include "../addition/addition.h"
5  #include "../subtraction/subtraction.h"
6
7  class Binops: public Addition, public Subtraction
8  {
9      friend class Addition;
10     friend class Subtraction;
11
12     private:
13         void binopsAdd(Operations const &rhs);
14         void binopsSub(Operations const &rhs);
15 };
16
17 #endif

```

addition/addition.h

```

1  #ifndef EX20_ADDITION_H
2  #define EX20_ADDITION_H
3
4  class Operations;
5
6  class Addition
7  {
8      friend Operations operator+(
9          Operations const &lhs, Operations const &rhs);    // 1
10     friend Operations operator+(
11         Operations &&lhs, Operations const &rhs);           // 2
12
13     public:
14         Operations &operator+=(Operations const &rhs) &;    // 1
15         Operations operator+=(Operations const &rhs) &&;    // 2
16 };
17
18 #endif

```

subtraction/subtraction.h

```
1 #ifndef EX20_SUBTRACTION_H
2 #define EX20_SUBTRACTION_H
3
4 class Operations;
5
6 class Subtraction
7 {
8     friend Operations operator-(
9         Operations const &lhs, Operations const &rhs);    // 1
10    friend Operations operator-(
11        Operations &&lhs, Operations const &rhs);          // 2
12
13    public:
14        Operations &operator--(Operations const &rhs) &; // 1
15        Operations operator--(Operations const &rhs) &&; // 2
16 };
17
18 #endif
```

Exercise 21

Learn to implement a class hierarchy using friends in the final derived class

Below we provide the implementations of the classes `Binops`, `Addition` and `Subtraction`. (Not the free binary operators, see the next exercise for those.)

binops/binops.ih

```
1 | #include "binops.h"
2 | #include "../operations/operations.h"
```

binops/binopsadd.cc

```
1 | #include "binops.ih"
2 |
3 | void Binops::binopsAdd(Operations const &rhs)
4 | {
5 |     static_cast<Operations &>(*this).add(rhs);
6 | }
```

binops/binopssub.cc

```
1 | #include "binops.ih"
2 |
3 | void Binops::binopsSub(Operations const &rhs)
4 | {
5 |     static_cast<Operations &>(*this).sub(rhs);
6 | }
```

addition/addition.ih

```
1 | #include "addition.h"
2 | #include "../operations/operations.h"
3 |
4 | #include <utility>
5 |
6 | using namespace std;
```

addition/operatoraddis1.cc

```
1 #include "addition.ih"
2
3 Operations &Addition::operator+=(Operations const &rhs) &
4 {
5     cout << "operatoraddis1 calls: ";
6     static_cast<Binops &>(*this).binopsAdd(rhs);
7     return static_cast<Operations &>(*this);
8 }
```

addition/operatoraddis2.cc

```
1 #include "addition.ih"
2
3 Operations Addition::operator+=(Operations const &rhs) &&
4 {
5     cout << "operatoraddis2 calls: ";
6     static_cast<Binops &>(*this).binopsAdd(rhs);
7     return move(static_cast<Operations &>(*this));
8 }
```

subtraction/subtraction.ih

```
1 #include "subtraction.h"
2 #include "../operations/operations.h"
3
4 #include <utility>
5
6 using namespace std;
```

subtraction/operatorsubis1.cc

```
1 #include "subtraction.ih"
2
3 Operations &Subtraction::operator-=(Operations const &rhs) &
4 {
5     cout << "operatorsubis1 calls: ";
6     static_cast<Binops &>(*this).binopsSub(rhs);
7 }
```

```
7 |     return static_cast<Operations &>(*this);
8 | }
```

subtraction/operatorsubis2.cc

```
1 | #include "subtraction.ih"
2 |
3 | Operations Subtraction::operator-=(Operations const &rhs) &&
4 | {
5 |     cout << "operatorsubis2 calls: ";
6 |     static_cast<Binops &>(*this).binopsSub(rhs);
7 |     return move(static_cast<Operations &>(*this));
8 | }
```

Exercise 22

Learn to use a class hierarchy using friends in the final derived class

Below we provide the remaining members of the classes `Binops`, `Addition` and `Subtraction`. The members `add` and `sub` are implemented inline, and therefore we provide the interface of `Operations` as well. After those listings, we provide a main function calling the different binary operators, and the output it produces. From that we see that the code works as intended.

addition/operatoradd1.cc

```
1 #include "addition.ih"
2
3 Operations operator+(Operations const &lhs, Operations const &rhs)
4 {
5     cout << "operatoradd1 calls: ";
6     Operations ret(lhs);
7     ret += rhs;
8     return ret;
9 }
```

addition/operatoradd2.cc

```
1 #include "addition.ih"
2
3 Operations operator+(Operations &&lhs, Operations const &rhs)
4 {
5     cout << "operatoradd2 calls: ";
6     Operations ret(move(lhs));
7     ret += rhs;
8     return ret;
9 }
```

subtraction/operatorsub1.cc

```
1 #include "subtraction.ih"
2
3 Operations operator-(Operations const &lhs, Operations const &rhs)
```

```

4 | {
5 |     cout << "operatorsub1 calls: ";
6 |     Operations ret(lhs);
7 |     ret -= rhs;
8 |     return ret;
9 | }

```

subtraction/operatorsub2.cc

```

1 | #include "subtraction.ih"
2 |
3 | Operations operator-(Operations &&lhs, Operations const &rhs)
4 | {
5 |     cout << "operatorsub2 calls: ";
6 |     Operations ret(move(lhs));
7 |     ret -= rhs;
8 |     return ret;
9 | }

```

operations/operations.h

```

1 | #ifndef EX20_OPERATIONS_H
2 | #define EX20_OPERATIONS_H
3 |
4 | #include <iostream>
5 | #include "../binops/binops.h"
6 |
7 | class Operations: public Binops
8 | {
9 |     friend Binops;
10 |
11 |     public:
12 |         Operations() = default;
13 |
14 |     private:
15 |         void add(Operations const &rhs);
16 |         void sub(Operations const &rhs);
17 | };
18 |

```



```

19 inline void Operations::add(Operations const &rhs)
20 {
21     std::cout << "addition\n";
22 }
23
24 inline void Operations::sub(Operations const &rhs)
25 {
26     std::cout << "subtraction\n";
27 }
28
29 #endif

```

main.cc

```

1 #include "operations/operations.h"
2
3 int main()
4 {
5     Operations obj;
6
7     obj += obj;
8     Operations{} += obj;
9     Operations obj2 = obj + obj;
10    obj2 = Operations{} + obj;
11
12    obj -= obj;
13    Operations{} -= obj;
14    Operations obj3 = obj - obj2;
15    obj3 = Operations{} - obj;
16 }

```

Output of main.cc

```

operatoraddis1 calls: addition
operatoraddis2 calls: addition
operatoradd1 calls: operatoraddis1 calls: addition
operatoradd2 calls: operatoraddis1 calls: addition
operatorsubis1 calls: subtraction
operatorsubis2 calls: subtraction

```

```
| operatorsub1 calls: operatorsubis1 calls: subtraction  
| operatorsub2 calls: operatorsubis1 calls: subtraction
```

Exercise 23

Learn to use a class hierarchy using friends in the final derived class

Exercise 24

Learn to initialize `string` objects with `new`

We used the following code,

main.ih

```
1 #include <string>
2 #include <iostream>
3
4 using namespace std;
5
6 string *factory(string const &str, size_t count);
```

main.cc

```
1 #include "main.ih"
2
3 int main()
4 {
5     string str = "test";
6     size_t count = 3;
7     string *sp = factory(hoi, count);
8     for (size_t idx = 0; idx != count; ++idx)
9         cout << sp[idx] << '\n';
10 }
```

factory.cc

```
1 #include "main.ih"
2
3 string *factory(string const &str, size_t count)
4 {
5     static string inputStr = str;          // made static s.t. Xstr has access
6
7     class Xstr: public string
8     {
9
```

```
10         public:
11             Xstr()
12             :
13                 string(inputStr)
14             {}
15     };
16
17     return new Xstr[count];
18 }
```