

# Programming in C/C++

## Exercises set two: advanced class templates

Christiaan Steenkist  
Jaime Betancor Valado  
Remco Bos

February 2, 2017

### Exercise 9, Needle fishing

We made a function that returns the place of the first template class in a haystack of classes.

#### Code listings

Listing 1: main.cc

```
1  #include "type.h"
2
3  int main()
4  {
5      cout <<
6          Type<int>::located << ' ' <<
7          Type<int, double>::located << ' ' <<
8          Type<int, int>::located << ' ' <<
9          Type<int, double, int>::located << ' ' <<
10         Type<int, double, int>::located << ' ' <<
11         Type<int, double, int, int, int>::located <<
12         '\n';
13 }
```

Listing 2: type.h

```
1  #ifndef TYPE_H
2  #define TYPE_H
```

```

3
4 #include <iostream>
5
6 using namespace std;
7
8 // TYPEIDX LOOPER
9 // Generic case: no Needle found
10 template <typename Needle, size_t index,
11         typename Other, typename ...Stack>
12 class TypeIdx : public TypeIdx<Needle,
13         index + 1, Stack...>
14 {};
15
16 // Exit case: Needle found
17 template <typename Needle, size_t index,
18         typename ...Stack>
19 class TypeIdx<Needle, index, Needle, Stack...>
20 {
21     public:
22         enum { located = index + 1 };
23 };
24
25 // Exit case: Needle found
26 template <typename Needle, size_t index>
27 class TypeIdx<Needle, index, Needle>
28 {
29     public:
30         enum { located = index + 1 };
31 };
32
33 // Exit case: No more hay
34 template <typename Needle, size_t index,
35         typename Other>
36 class TypeIdx<Needle, index, Other>
37 {
38     public:
39         enum { located = 0 };
40 };
41
42 // TYPE STARTER

```

```

43 // Starter for TypeIdx and looping
44 template <typename Needle, typename ...Stack>
45 class Type : public TypeIdx<Needle, 0, Stack...>
46 {};
47
48 // Starter for empty haystack
49 template <typename Needle>
50 class Type<Needle>
51 {
52     public:
53         enum { located = 0};
54 };
55
56 #endif

```

## Exercise 10, Needle fishing with nested class

We changed exercise 9, such that it now uses a nested helper class

### Code listings

Listing 3: main.cc

```

1  #include "type.h"
2
3  int main()
4  {
5      cout <<
6          Type<int>::located << ' ' <<
7          Type<int, double>::located << ' ' <<
8          Type<int, int>::located << ' ' <<
9          Type<int, double, int>::located << ' ' <<
10         Type<int, double, int>::located << ' ' <<
11         Type<int, double, int, int, int>::located <<
12         '\n';
13 }

```

Listing 4: type.h

```

1  #ifndef TYPE_H
2  #define TYPE_H

```

```

3
4 #include <iostream>
5
6 using namespace std;
7
8 // Starter for the needlehunt
9 template <typename Needle, typename ...Stack>
10 class Type
11 {
12     // Generic case: no Needle found
13     template <size_t index, typename Other,
14             typename ...Stacker>
15     struct TypeIdx
16     {
17     public:
18         enum {located = TypeIdx<index + 1,
19             Stacker...>::located};
20     };
21
22     // Exit case: Needle found
23     template <size_t index, typename ...Stacker>
24     struct TypeIdx<index, Needle, Stacker...>
25     {
26     public:
27         enum { located = index + 1 };
28     };
29
30     // Exit case: Needle found
31     template <size_t index>
32     struct TypeIdx<index, Needle>
33     {
34     public:
35         enum { located = index + 1};
36     };
37
38     // Exit case: No more hay
39     template <size_t index, typename Other>
40     struct TypeIdx<index, Other>
41     {
42     public:

```

```

43     enum { located = 0 };
44 };
45
46 public:
47     enum { located = TypeIdx<0, Stack...>::located};
48 };
49
50 // Starter for empty haystack
51 template <typename Needle>
52 class Type<Needle>
53 {
54 public:
55     enum { located = 0};
56 };
57
58 #endif

```

## Exercise 13, Binary operators

We made a class that overloads binary operators

### Code listings

Listing 5: main.cc

```

1  #include "main.ih"
2
3  int main(int argc, char **argv)
4  {
5      Arithmetic<int> jart;
6      Arithmetic<double> dart;
7      //Arithmetic<size_t> illegal;
8
9      jart = 10;
10     dart = 10;
11     //illegal = 10;
12
13     Arithmetic<double> dart2(dart);
14     dart = 13;
15
16     cout << jart.value() << '\n';

```

```

17     cout << dart.value() << '\n';
18     cout << dart2.value() << '\n';
19 }

```

Listing 6: main.ih

```

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

```

Listing 7: adder.add.cc

```

1 #include "adder.i"
2
3 void Adder::add(Adder const &rhs)
4 {
5     d_value += rhs.value();
6 }

```

Listing 8: adder.h

```

1 #ifndef ADDER_H
2 #define ADDER_H
3
4 #include <string>
5
6 #include "binopsbase.h"
7
8 class Adder: public BinopsBase<Adder, '+'>
9 {
10     friend BinopsBase<Adder, '+'>;
11
12     std::string d_value;
13
14     public:
15         Adder &operator=(std::string const &rhs);
16         std::string const &value() const;
17
18     private:
19         void add(Adder const &rhs);

```

```

20 };
21
22 #endif

```

Listing 9: adder.ih

```

1 #include "adder.h"

```

Listing 10: adder.value.cc

```

1 #include "adder.ih"
2
3 std::string const &Adder::value() const
4 {
5     return d_value;
6 }

```

Listing 11: arithmetic.h

```

1 #ifndef ARITHMETIC_H
2 #define ARITHMETIC_H
3
4 #include <cstring>
5 #include <string>
6
7 #include "binopsbase.h"
8
9 template <typename T>
10 class Arithmetic: public BinopsBase<Arithmetic<T>>
11 {};
12
13 template <>
14 class Arithmetic<int>:
15     public BinopsBase<Arithmetic<int>>
16 {
17     friend BinopsBase<Arithmetic<int>>;
18
19     int d_value = 0;
20
21 public:
22     Arithmetic<int>() = default;
23     Arithmetic<int>(Arithmetic<int> const &rhs)

```

```

24     :
25         d_value(rhs.d_value)
26     {}
27     Arithmetic<int>(Arithmetic<int> &&rhs)
28     {
29         swap(rhs);
30     }
31
32     Arithmetic<int> &operator=(int const &rhs)
33     {
34         d_value = rhs;
35         return *this;
36     }
37
38     int const &value() const
39     {
40         return d_value;
41     }
42
43     void swap(Arithmetic<int> &other)
44     {
45         char step[sizeof(Arithmetic<int>)];
46         std::memcpy(step, this,
47                     sizeof(Arithmetic<int>));
48         std::memcpy(&other, step,
49                     sizeof(Arithmetic<int>));
50         std::memcpy(this, &other,
51                     sizeof(Arithmetic<int>));
52     }
53
54     private:
55         void add(Arithmetic<int> const &rhs)
56         {
57             d_value += rhs.value();
58         }
59 };
60
61 template <>
62 class Arithmetic<double>:
63     public BinopsBase<Arithmetic<double>>

```



```

64 {
65     friend BinopsBase<Arithmetic<double>>;
66
67     double d_value = 0;
68
69     public:
70         Arithmetic<double>() = default;
71         Arithmetic<double>(Arithmetic<double> const &rhs)
72         :
73             d_value(rhs.d_value)
74         {}
75         Arithmetic<double>(Arithmetic<double> &&rhs)
76         {
77             swap(rhs);
78         }
79
80         Arithmetic<double> &operator=(double const &rhs)
81         {
82             d_value = rhs;
83             return *this;
84         }
85
86         double const &value() const
87         {
88             return d_value;
89         }
90
91         void swap(Arithmetic<double> &other)
92         {
93             char step[sizeof(Arithmetic<double>)];
94             std::memcpy(step, this,
95                 sizeof(Arithmetic<double>));
96             std::memcpy(&other, step,
97                 sizeof(Arithmetic<double>));
98             std::memcpy(this, &other,
99                 sizeof(Arithmetic<double>));
100         }
101
102     private:
103         void add(Arithmetic<double> const &rhs)

```

```

104     {
105         d_value += rhs.value();
106     }
107 };
108
109 #endif

```

Listing 12: arithmetic.ih

```

1 #include "arithmetic.h"

```

Listing 13: binopsbase.h

```

1 #ifndef BINOPSBASE_H
2 #define BINOPSBASE_H
3
4 template <typename Derived, int ...operators>
5 class BinopsBase
6 {
7     friend Derived &operator+(Derived &lhs,
8         Derived const &rhs);
9     friend Derived &operator*(Derived &lhs,
10         Derived const &rhs);
11
12     void addWrap(Derived const &rhs)
13     {
14         Derived::add(rhs);
15     }
16
17     void mulWrap(Derived const &rhs)
18     {
19         Derived::mul(rhs);
20     }
21 };
22
23 #endif

```

Listing 14: binopsbase.ih

```

1 #include "binopsbase.h"

```

## Exercise 14, Generic variadic template

We changed the class BinopsBase to a variadic template class using a set of int argument.

### Code listings

Listing 15: main.cc

```
1  #include "main.ih"
2
3  int main(int argc, char **argv)
4  {
5      Arithmetic<int> jart;
6      Arithmetic<double> dart;
7      //Arithmetic<size_t> illegal;
8
9      jart = 10;
10     dart = 10;
11     //illegal = 10;
12
13     Arithmetic<double> dart2(dart);
14     dart = 13;
15
16     cout << jart.value() << '\n';
17     cout << dart.value() << '\n';
18     cout << dart2.value() << '\n';
19 }
```

Listing 16: main.ih

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

Listing 17: operations.h

```
1  #ifndef OPERATIONS_H
2  #define OPERATIONS_H
3
```

```

4  template <typename Base, typename Derived>
5  class Add
6  {};
7
8  template <typename Base, typename Derived>
9  class Mul
10 {};
11
12 #endif

```

Listing 18: operations.ih

```

1  #include "operations.h"

```

Listing 19: binopsbase.h

```

1  #ifndef BINOPSBASE_H
2  #define BINOPSBASE_H
3
4  #include "operations.h"
5
6  template <typename Binops,
7          typename Derived, int ...operators>
8  class BinopsBase0
9  {};
10
11 template <typename Binops, typename Derived>
12 class BinopsBase0<Binops, Derived, 0>
13 {};
14
15 template <typename Derived, int ...operators>
16 class BinopsBase : public BinopsBase0<BinopsBase<
17     Derived, operators..., Derived, operators...>
18 {
19     friend Derived &operator+(Derived &lhs,
20         Derived const &rhs);
21     friend Derived &operator*(Derived &lhs,
22         Derived const &rhs);
23
24     friend Add<BinopsBase<Derived, operators...,
25         Derived>;

```

```

26     friend Mul<BinopsBase<Derived, operators...>,
27           Derived>;
28
29     void addWrap(Derived const &rhs)
30     {
31         Derived::add(rhs);
32     }
33
34     void mulWrap(Derived const &rhs)
35     {
36         Derived::mul(rhs);
37     }
38 };
39
40 #endif

```

Listing 20: binopsbase.ih

```

1 #include "binopsbase.h"

```