Exercises week 1: Function Templates

Klaas Isaac Bijlsma s2394480
David Vroom s2309939

February 14, 2018

Exercise 1

Show that templates don't result in 'code bloat'

A function template `add` and a union `PointerUnion` were defined in separate header files. We use this union to print the address of a function. There are two source files, one for `fun` and one for `main`. The function `fun`, which includes `add.h`, instantiates `add` for `int`s and prints its address. Then, in `main` the same happens and `fun` is called. When the two source files of `fun` and `main` are compiled to object modules, they both contain an instantiation of `add`. Then they are linked to obtain an executable. The output of this executable gives two identical addresses, which means that only one instantiation of `add` is present. So the linker prevents 'code bloat'.

add.h

```
1 template <typename Type>
2
3 Type add(Type const &lhs, Type const &rhs)
4 {
5     return lhs + rhs;
6 }
```

pointerunion.h

```
1 union PointerUnion
2 {
```

```
3       int (*fp)(int const &, int const &);
4       void *vp;
5   };
```

main.cc

```
1   #include <iostream>
2   #include "add.h"
3   #include "pointerunion.h"
4
5   void fun();
6
7   int main()
8   {
9       PointerUnion pu = { add };
10
11      fun();
12      std::cout << pu.vp << '\n';
13  }
```

fun.cc

```
1   #include <iostream>
2   #include "add.h"
3   #include "pointerunion.h"
4
5   void fun()
6   {
7       PointerUnion pu = { add };
8
9       std::cout << pu.vp << '\n';
10  }
```

Exercise 2

Learn to embed a function template in a function template

We used the following code,

as.h

```
1  template <typename Type1, typename Type2>
2
3  Type1 as(Type2 const &value)
4  {
5      return static_cast<Type1>(value);
6  }
```

main.cc

```
1  #include <iostream>
2  #include "as.h"
3
4  using namespace std;
5
6  int main()
7  {
8      int chVal = 'X';
9
10     cout << as<char>(chVal) << '\n';
11 }
```

Exercise 3

Exercise 4

Exercise 5