Exercises for the Lecture Series

"Object-Oriented Programming for Scientific Computing"

Dr. Ole Klein
ole.klein@iwr.uni-heidelberg.de

To be handed in on 30. 05. 2017 before the lecture

Exercise 1 Empty Base Class Optimization

In this exercice we will examine how much space empty C++ classes and their derived classes use.

First create a class `Empty`, which doesn't possess any attributes. Use inheritance to derive a class `EmptyDerived` from `Empty`, which as well lacks attributes. Also create a class `NonEmpty`, which is again derived from `Empty`, but features a single `char` as attribute.

What is the size of objects of the classes `Empty`, `EmptyDerived` and `NonEmpty` according to the **`sizeof`** operator? (By the way: This behavior is called empty base class optimization.)

To understand why the size of empty classes (according to standard) is as observed, consider the following class

```
struct Composite
{
  Empty a;
  int  b;
};
```

What would happen if `Empty` wouldn't claim any storage space? Let `c` be an object of the class `Composite`, which addresses would the attributes `c.a` and `c.b` have in this case?

Determine the size of objects belonging to class `Composite`! How does it change when you change the type of the attribute `b` to `char`? (Try it!)

6 Points

Exercise 2 Inheritance and Composition

Write a class `NumVector`, which represents a numerical vector. The size of the vector has to be fixed. Read and write access to the individual entries of the vector should be granted by means of the brackets operator. In addition, the class should have the method **`double`** `norm()` that calculates the Euclidean norm of the vector. Other methods should not be visible. In particular, a `NumVector` should not be a `std::vector<double>`!

Design two implementations of this class, both of which store the values in a `std::vector<double>`. The first implementation has to be derived from `std::vector<double>` and the second one should not use inheritance. Use two different header files for this, and only include one of them at a time when testing with the programs below.

Test your implementation as follows: The classes should work correctly with the following **`main`** program.

```
#include<vector>
#include<iostream>
#include<cmath>
int main(){
```

```
  NumVector v(3);
  v[0]=1; v[1]=3, v[2]=4;
  const NumVector& w=v;
  std::cout<<"norm is "<<w.norm()<<std::endl;
  std::cout<<"vector is ["<<w[0]<<", "<<w[1]<<", "<<w[2]<<"]"<<std::endl;
}
```

The following program must not compile though!

```
#include<vector>
#include<iostream>
#include<cmath>

void vectorTest(std::vector<double>& v){}

int main(){
  NumVector v(3);
  v.resize(2); // must be hidden, just like other methods of std::vector!
  vectorTest(v); // this should also lead to a compiler error!
}
```

10 Points

EXERCISE 3 INHERITANCE AND FUNCTIONS

Find the errors in the following program. What is the output after the incorrect lines are removed, and why?

```
1  #include<iostream>
2
3  class A{
4  public:
5    void foo() const { std::cout<<"A::foo"<<std::endl; }
6  };
7
8  class B : public A{
9  public:
10   void foo() { std::cout<<"B::foo"<<std::endl; }
11 };
12
13 class C : private B{
14 public:
15   void bar() { foo();}
16 };
17
18 void test(const A& a){ a.foo(); }
19
20 int main(){
21   A a; B b; C c;
22   a.foo();
23   b.foo();
24   test(b);
25   c.bar();
26   test(c);
27 }
```

4 Points