C++ Programming

Exercise Sheet 8
Secure Software Engineering Group
Philipp Schubert
philipp.schubert@upb.de

June 11, 2021

Solutions to this sheet are due on 18.06.2021 at 16:00. Please hand-in a digital version of your answers via PANDA at https://panda.uni-paderborn.de/course/view.php?id=22691.
Note: If you copy text or code elements from other sources, clearly mark those elements and state the source. Copying solutions from other students is prohibited.

This exercise sheet is all about object oriented programming (OOP). You can achieve 16 points in total.

Exercise 1.
Consider the following code:

```
#include <iostream>

class base {
 public:
  virtual ~base() = default;
  virtual void whoAmI() { std::cout << "I am base\n"; }
};
```

a) Define two types **derived_one** and **derived_two** using the **class** keyword that both inherit from **base**. (1 P.)

b) In each of the derived classes override the virtual **whoAmI()** function member such that it prints the name of the derived class. (1 P.)

c) Why is it a good idea to explicitly specify functions that override a virtual function with the keyword **override**? Describe a scenario where one gets into huge trouble when not having specified overriding functions as **override**! (2 P.)

Exercise 2.

Consider the following two interfaces:

```
#include <iostream>

struct greetings {
  virtual ~greetings() = default;
  virtual void say_hello() = 0;
  virtual void say_goodbye() = 0;
};

struct politeness {
  virtual ~politeness() = default;
  virtual void say_please() = 0;
  virtual void say_thanks() = 0;
  virtual void say_your_welcome() = 0;
};
```

Define a class **speaker** that implements both of the above interfaces. All interface functions should be implemented by writing an "adequate message" to the command line. Test your class **speaker** by creating an instance of that very class and calling all of its member functions. (2 P.)

Exercise 3.

Consider the following **container** interface:

```
class container {
 public:
  virtual ~container() = default;
  virtual double& operator[] (size_t idx) = 0;
  virtual const double& operator[] (size_t idx) const = 0;
  virtual size_t size() const = 0;
};
```

a) Define a class **vec** that implements the **container** interface. Use a member variable of type **std::vector<double>** to store the elements in your **vec** type. Additionally, provide a constructor **vec(size_t size)** that initializes the member variable such that it is capable of holding **size** elements. (4 P.)

b) Define another class **lst** that also implements the **container** interface. But this time, use a member variable of type **std::list<double>** to store the elements in your **lst** type. Also provide a constructor **lst(size_t size)** that initializes the member such that it is able to store **size** elements. (Hint: when implementing **operator[]** for your list wrapper, the function **std::advance** may come in handy.) (4 P.)

c) Observe the code shown below. The functions **fill_container()** and **sum_container()** can operate on any type that implements the **container** interface. Create a variable of your **vec** and a variable of your **lst** type such that they can both store 10 **double** elements. Then, call **fill_container()** and **sum_content()** with each of those variables. You should obtain 55, as a result, in both cases. (2 P.)

```
void fill_container(container& c) {
  for (size_t i = 0; i < c.size(); ++i) {
    c[i] = i + 1;
  }
}
```

```
double sum_container(const container& c) {
  double sum = 0;
  for (size_t i = 0; i < c.size(); ++i) {
    sum += c[i];
  }
  return sum;
}
```

Exercise 4.

This is an optional exercise. Consider the following code:

```
#include <iostream>

template <class T> class base {
protected:
  T base_value;

public:
  base(T t) : base_value(t) {}
};

template <class T> class derived : public base<T> {
private:
  T derived_value;

public:
  derived(T t, T u) : base<T>(u), derived_value(t) {}
  void printValues() {
    std::cout << base_value << '\n';
    std::cout << derived_value << '\n';
  }
};

int main() {
  derived<int> d(20, 10);
  d.printValues();
  return 0;
}
```

Try to compile and execute the code. The compilation should result in an error. Can you fix the error? (Hint: you may wish to precisely google for the right terms to find the solution.)
(0 P.)

Exercise 5.

Additional material: I highly recommend to watch the recording of the talk "Intro to the C++ Object Model", by Richard Powell (CppCon 2015) `https://youtu.be/iLiDezv_Frk` to deepen and extend your knowledge about C++'s object model. (0 P.)