# Examination
# C++ Programming

### 2016–03–19, 8.00–13.00

Aid at the exam: one C++ book. The copies of the lecture slides are *not* allowed.

You must show that you know C++ and that you can use the C++ standard classes. "C solutions" don't give any points, even if they are correct.

Assessment (preliminary): the questions give $16 + 6 + 2 + 18 + 8 = 50$ points. You need 25 points for a passing grade (3/25, 4/33, 5/42).

---

1. The following class is given:

```
class A {
public:
    A(size_t sz = 10) : storage(new int[size = sz]) {}
    int operator[](size_t pos) const { return storage[pos]; }
    int& operator[](size_t pos) { return storage[pos]; }
private:
    int* storage; size_t size;
};
```

   a) When you try to use the class in the function below you get a compilation error. Where, and how do you best correct the error? Answer with the missing code and state where in the program it should be inserted.

```
void print(const string& msg, A a) {
    cout << msg << " " << a << endl;
}
void f() {
    A a1;
    for (size_t i = 0; i != 10; ++i) { a1[i] = i + 1; }
    print("This is a1: ", a1);
    A a2;
    a2 = a1;
    print("This is a2: ", a2);
}
```

   b) Once the problem above is corrected you are left with a memory leak. What additional constructs are needed in the class to eliminate the risk for memory leaks and why are the constructs needed? Implement the missing constructs and shortly state the reason they are needed.

   c) Before C++11 programmers often referred to a rule called "the rule of three". Now, this rule has been extended and we are now talking about "the rule of five". What does "the rule of five" state and what was the reason behind the addition to the rule?

   d) It is in most cases recommended to make destructors in C++ virtual. Why? Answer by giving an example which would otherwise be erroneous.

2. The following classes and functions are given:

```
struct Foo{
    virtual void print() const {std::cout << "Foo" << std::endl;}
};
struct Bar : Foo {
    virtual void print() const override {std::cout << "Bar" << std::endl;}
};
struct Qux : Bar {
    virtual void print() const override {std::cout << "Qux" << std::endl;}
};
void print1(const Foo* f)
{
    f->print();
}
void print2(const Foo& f)
{
    f.print();
}
void print3(Foo f)
{
    f.print();
}

int main() {
    Foo* a =  new Bar;
    Bar& b = *new Qux;
    Bar  c = *new Qux;

    print1(a);
    print1(&b);
    print1(&c);
    std::cout << std::endl;

    print2(*a);
    print2(b);
    print2(c);
    std::cout << std::endl;

    print3(*a);
    print3(b);
    print3(c);
    std::cout << std::endl;
}
```

   The program can be compiled and run without execution error.
   What is printed when `main()` is executed?

3. In C++ programs you sometimes find something called virtual inheritance. Explain what virtual inheritance is and why it is needed. Also illustrate how you specify that a class should inherit virtually.

4. 1-wire is a field bus standard developed by Dallas Semiconductor for communication with small, simple, and cheap units such as temperature sensors and similar. A 1-wire network can either be connected directly to a computer using a special adapter or one can use a so called 1-wire-to-ethernet bridge. Via such a bridge, sensors on the 1-wire network can be accessed from computers on the ethernet network. An example of such a bridge is OW-SERVER from Embedded Data Systems[1]. This bridge continuously scans the 1-wire network for sensors, reads their current input values, and stores the values in the memory of the bridge. These measurements can then be accessed from the ethernet side via a web server from which thecollected measurements can be accessed in the form of an XML file.

   Appendix A (see the end of the exam) shows how such an XML file can look like for a bridge with four connected 1-wire temperature sensors. For each sensor you find a section in the file starting with ”<owd_XXXXXX” there ”XXXXXX” is the model number for the sensor, The section ends with a corresponding ”</owd_XXXXXX>". Inbetween these tags we find two fields of interest. The first one gives us the unique identifier for the sensor. This identifier is found between the tags ”<ROMId>” and ”</ROMId>”. The other field is the latest temperature measurement from the sensor. You find this between the tags ”<Temperature Units=”Centigrade”>” and ”</Temperature>”. For the first sensor in the example, the identifier is thus ”630000041C430228” and the temperature is 5,1875°C.

   Your task is to implement a class `Sensors` using which it is possible to read temperature measurements from one or several bridges (typically located in different buildings) of the type OW-SERVER and continuously store and present the latest measurements from individual or all connected sensors.

   The class `Sensors` shall have the following public interface:

```
class Sensors {
public:
    // Constructor
    Sensors();

    // Connects to the OW-SERVER at the address given by url. Reads the current
    // XML data and stores information about the current temperature values.
    // Any previous values for the sensors in the XML file are overwritten.
    // Does not throw any exceptions. Any network errors are ignored (and no
    // data is read).
    void update(std::string url);

    // Returns the latest temperature read from the sensor given by id. Throws a
    // runtime_exception if there is no data available for the given sensor.
    double getTemp(const std::string& id) const;

    // Prints a list of all sensors (id and latest temperature value) sorted by
    // sensor id.
    void print() const;
};
```

   A simple test program for the class could look like this:

```
int main() {
        Sensors sensors;
        sensors.update("http://owserver1.lth.se/detail.xml");
        sensors.update("http://owserver2.lth.se/detail.xml");
        sensors.update("http://nonexistentserver.lth.se/detail.xml");
        std::cout << sensors.getTemp("0B000801848EAA10") << std::endl;
        sensors.print();
        std::cout << sensors.getTemp("XXXXXXXXXXXXXXX") << std::endl;
}
```

   The corresponding output from the test program could look like this (see next page):

[1] http://www.embeddeddatasystems.com/OW-SERVER-1-Wire-to-Ethernet-Server-Revision-2_p_152.html

```
22.5
06000006AE3F4528 20.1875
0B000801848EAA10 22.5
0C000BE765C89788 5.5
460008018487C610 20.5
480000067BC58721 19.5
630000041C430228 5.1875
libc++abi.dylib: terminating with uncaught exception of type std::runtime_error:
Unknown sensor.
Abort trap: 6
```

To your aid you is the following class `URLstream` which is a subclass to `istream` with which you can connect to an OW-SERVER and read the contents of the XML file. You are *NOT* supposed to implement this class.

```
// Opens a network connection to a document at a web server. Once open, the
// contents of the document can be read using standard stream operations.
class URLstream : public std::istream {
public:
    // Creates a connection to the web document referenced by url.
    // Throws std::ios_base::failure on network error.
    URLstream(std::string url);
};
```

Use the data structures, classes, and algorithms from the C++ standard library where appropriate.

5. The STL function `remove_copy_if` copies elements from one range to another range, except for those elements meeting a given condition. In the following example, all numbers larger than or equal to a number read from standard input (`cin`) are copied to the vector `b` and are subsequently written to standard output:

```
int c = 0;

bool ltc(int x) { return x < c; }

int main() {
    int a[] = { 5, 10, -3, 2, -8, 11 };
    const size_t SIZE = sizeof(a) / sizeof(int);
    int b[SIZE];
    cin >> c;
    int* last = remove_copy_if(a, a + SIZE, b, ltc);
    int* first = b;
    while (first != last) { // prints 5 10 2 11
        cout << *first++ << endl;
    }
}
```

   a) Unfortunately, the solution requires that the number read from standard input is stored in a global variable in order to be available in the function `ltc`. However, you can do without the `ltc` function and the global variable if you use a so called lambda function instead. How would the call to `remove_copy_if` look like in this case (we assume that `c` is now a local variable in `main`)?

   b) If all you want to do is to print the numbers larger than or equal to than the number read from standard input, it is unnecessary to first copy them to the vector `b`. By modifying the call to `remove_copy_if` we can print the numbers directly. How would the call to `remove_copy_if` look like in this case?

   c) Implement the function `remove_copy_if`. The function should of course accept elements of arbitrary types and be able to fetch them from a `vector`, `deque`, or any other STL container (i.e.,

the first three parameters should be iterators).

## Appendix A: OW-SERVER XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<Devices-Detail-Response xmlns="http://www.embeddeddatasystems.com/schema/owserver"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<PollCount>56610</PollCount>
<DevicesConnected>4</DevicesConnected>
<LoopTime>3.951</LoopTime>
<DevicesConnectedChannel1>1</DevicesConnectedChannel1>
<DevicesConnectedChannel2>2</DevicesConnectedChannel2>
<DevicesConnectedChannel3>1</DevicesConnectedChannel3>
<DataErrorsChannel1>0</DataErrorsChannel1>
<DataErrorsChannel2>2</DataErrorsChannel2>
<DataErrorsChannel3>1</DataErrorsChannel3>
<VoltageChannel1>4.60</VoltageChannel1>
<VoltageChannel2>4.59</VoltageChannel2>
<VoltageChannel3>4.60</VoltageChannel3>
<VoltagePower>4.83</VoltagePower>
<DeviceName>OWServer_v2-Enet</DeviceName>
<HostName>EDSOWSERVER2</HostName>
<MACAddress>00:05:3A:35:9C:4E</MACAddress>
<owd_DS18B20 Description="Programmable resolution thermometer">
<Name>DS18B20</Name>
<Family>28</Family>
<ROMId>630000041C430228</ROMId>
<Health>7</Health>
<Channel>1</Channel>
<RawData>53004B467FFF0D10D4000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000</RawData>
<PrimaryValue>5.1875 Deg C</PrimaryValue>
<Temperature Units="Centigrade">5.1875</Temperature>
<UserByte1 Writable="True">75</UserByte1>
<UserByte2 Writable="True">70</UserByte2>
<Resolution>12</Resolution>
<PowerSource>0</PowerSource>
</owd_DS18B20>
<owd_DS18S20 Description="Parasite power thermometer">
<Name>DS18S20</Name>
<Family>10</Family>
<ROMId>460008018487C610</ROMId>
<Health>7</Health>
<Channel>2</Channel>
<RawData>29004B46FFFF0310C8000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000</RawData>
<PrimaryValue>20.5000 Deg C</PrimaryValue>
<Temperature Units="Centigrade">20.5000</Temperature>
<UserByte1 Writable="True">75</UserByte1>
<UserByte2 Writable="True">70</UserByte2>
</owd_DS18S20>
<owd_DS18B20 Description="Programmable resolution thermometer">
<Name>DS18B20</Name>
<Family>28</Family>
<ROMId>06000006AE3F4528</ROMId>
<Health>7</Health>
<Channel>2</Channel>
```

```
<RawData>43014B467FFF0D10BD0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000</RawData>
<PrimaryValue>20.1875 Deg C</PrimaryValue>
<Temperature Units="Centigrade">20.1875</Temperature>
<UserByte1 Writable="True">75</UserByte1>
<UserByte2 Writable="True">70</UserByte2>
<Resolution>12</Resolution>
<PowerSource>0</PowerSource>
</owd_DS18B20>
<owd_DS18S20 Description="Parasite power thermometer">
<Name>DS18S20</Name>
<Family>10</Family>
<ROMId>0B000801848EAA10</ROMId>
<Health>7</Health>
<Channel>3</Channel>
<RawData>2D004B46FFFF0610220000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000</RawData>
<PrimaryValue>22.5000 Deg C</PrimaryValue>
<Temperature Units="Centigrade">22.5000</Temperature>
<UserByte1 Writable="True">75</UserByte1>
<UserByte2 Writable="True">70</UserByte2>
</owd_DS18S20>
</Devices-Detail-Response>
```