Problem 1

An IDL description

```
module Aufgabe2{
  interface Server{
    string reverse(in string message);
  };
};
```

and an object reference

IOR:000000000000001849444C3A4175666761626532 2F5365727665723A312E300000000000100000000000000280001000000000000F3134312E38392E3232342E3133310000162E000000000000086175666761626532

corbaloc::141.89.224.131:5678/aufgabe2

are given. Use these to implement a C++ client that calls the operation `reverse`.

First I had to choose a suitable CORBA runtime environment. I decided to use the light-weight implementation MICO that is freely available under the GPL. Unfortunately it took more than 110 (!) minutes to initially build the library on my parent's 233 MHz computer. Nevertheless version 2.3.9 turned out to be a Visual-Studio-.Net-compliant one which is not the case for all CORBAs and worth a lot (at least to me).

A second problem was caused by the Hasso-Plattner-institute's firewall: it blocked any access to the CORBA service running on the given server. Due to the "closed" nature of Windows even the Python source code provided by Mr. von Löwis was not of use to me. Therefore I finally decided to write both a server and a client.

MICO comes with its own IDL compiler that generates all the stubs and proxies you need. Then, my "duty" was to write a server and client that utilize these files by deriving from the generated classes and bringing the pure virtual functions to life.

Because of my decision to write my own server, the given object reference must be replaced by a new one that I obtain directly from the server while it starts up: it creates a file called ior.txt that contains the required IOR. The client then accesses this file, reads the IOR and contacts the server (see figure 1).

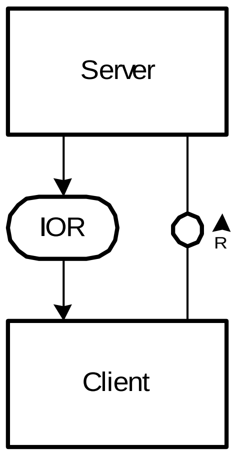

Figure 1: Static structure

My "private" IOR is:

```
IOR:010000001800000049444c3a4175666761626532 2f5365727665723a312e30000200000
000000000240000000010100000a0000003132372e302e302e3100d5100c000000424f417f00
000100000ec0010100000002400000000100000000100000000100000014000000001000000001000
1000000000000901010000000000
```

Seen from a technical point of view, the client has to perform some more steps:

1. Initialize the ORB.
2. I use the BOA approach, thus the BOA itself needs to be initialized, too.
3. Read the object reference from ior.txt.
4. Create a local CORBA object (a proxy) using the object reference.
5. Cast it by calling the `_narrow` operation.
6. Use the local CORBA object to call the remote `reverse` operation.

The code given below just follows these instructions:

client.cpp:

```
// ///////////////////////////////////////////////////////////
// Lecture on the CORBA Component Model, summer term 2003
// Assignment 2, Stephan Brumme, 702544
//
// CORBA client that class a remote "reverse" operation
//

// save and display IOR
#include <iostream>
#include <fstream>

// IDL compiler generated a stub
#include "Aufgabe2.h"
using namespace Aufgabe2;


int main(int argc, char* argv[])
{
      // initialize CORBA and its BOA components
      CORBA::ORB_var orb = CORBA::ORB_init(argc, argv, "mico-local-orb");
      CORBA::BOA_var boa = orb->BOA_init(argc, argv, "mico-local-boa");

      // read IOR
      char reference[1000];
      std::ifstream ior("ior.txt");
      ior >> reference;
      ior.close();
      std::cout << reference << std::endl;

      // create and cast the remote object
      CORBA::Object_var obj = orb->string_to_object(reference);
      Server_var remote = Server::_narrow(obj);

      // invoke the "reverse" operation
      std::cout << remote->reverse("Corba Component Model") << std::endl;

      return 0;
}
```

The server code does not differ a lot. Beside providing the source code for the `reverse` operation it needs to:

1. Initialize the CORBA library.
2. Initialize BOA.
3. Create a new server object (an object of the class that publishes the `reverse` operation, namely `Server_skel`).
4. Get the object reference (known as IOR) and write it out to ior.txt.
5. Wait for incoming requests.
6. Shutdown the ORB.

server.cpp:

```
// ////////////////////////////////////////////////////////////
// Lecture on the CORBA Component Model, summer term 2003
// Assignment 2, Stephan Brumme, 702544
//
// CORBA server that provides an operation to reverse
// a string
//

// save and display IOR
#include <iostream>
#include <fstream>

// IDL compiler generated a skeleton
#include "Aufgabe2.h"
using namespace Aufgabe2;


// the server
class Server_Impl : public virtual Server_skel
{
public:
      // IDL: string reverse(in string message);
      char* reverse(const char* message);
};


// reverse a given message
char* Server_Impl::reverse(const char* message)
{
      // some nice info
      std::cout << "Invoked by client, incoming: '" << message << "'";

      // create new string
      unsigned int length = strlen(message);
      char* result = CORBA::string_alloc(length);
      if (result == NULL)
            return result;

      // zero-terminated !
      result[length] = 0;

      // reverse the message
      for (unsigned i=0; i<length; i++)
            result[i] = message[length-i-1];
```

```
        // again some info
        std::cout << " - returning: '" << result << "'" << std::endl;
        // done !
        return result;
}



int main(int argc, char* argv[])
{
        // initialize CORBA and its BOA components
        CORBA::ORB_var orb = CORBA::ORB_init(argc, argv, "mico-local-orb");
        CORBA::BOA_var boa = orb->BOA_init(argc, argv, "mico-local-boa");

        // register object
        Server_Impl* server = new Server_Impl;
        CORBA::String_var reference = orb->object_to_string(server);

        // write out the object's IOR
        std::cout << reference << std::endl;
        std::ofstream ior("ior.txt");
        ior << reference;
        ior.close();

        // wait for requests
        boa->impl_is_ready(CORBA::ImplementationDef::_nil());
        orb->run();

        // game over
        CORBA::release(server);
        return 0;
}
```