Advanced Programming 2022.04.14 Exam Solution

Advanced Programming (IN1503) (Technische Universität München)

Chair of Scientific Computing
Department of Informatics
Technical University of Munich

Personal sticker

S0018

Compliance to the code of conduct
I hereby assure that I solve and submit this exam myself under my own name by only using the allowed tools listed below.

Signature or full name if no pen input available

Advanced Programming

Exam:	IN1503 / Retake	**Date:**	Thursday 14th April, 2022
Examiner:	Prof. Hans-Joachim Bungartz	**Time:**	14:15 – 15:30

Working instructions

- You have to **submit before the end of the submission period**; the upload period concerns only the technical transmission of an already submitted file. We will not accept any submissions after this time, but you can submit multiple times before that. There will be no approaching time announcement.
- Even though problems 2-4 follow the same theme, they are independent and can be solved in any order. Often, subproblems are also independently solvable, so make sure to try everything.
- You do not need to specify required header inclusions or the main function, unless explicitly asked.
- Answers are only accepted if the solution approach is documented. Give a reason for each answer unless explicitly stated otherwise in the respective subproblem.
- Do not write with red or green colors nor use pencils.
- Do not use comments or notes to write your answers. This will not be visible after submission.
- Do not forget to **save** the annotated PDF file. Verify that the annotations are visible in the submission overview.
- This is an “open book” exam. You are free to use any learning material.
- Communication with other people during the examination is **strictly prohibited**. We may invite you for a follow-up interview to justify this, if deemed needed.
- If you run into technical issues, we will be available in the usual lecture BBB room, where you can send us a short private message and we will contact you: `https://bbb.in.tum.de/ger-f3u-4w6`. We cannot answer any topic-related questions. In case of doubt, write your assumptions and continue.

Problems

1. Warming up (9 credits) 3
2. Data types, Functions, STL (8 credits) 5
3. Object-oriented Programming (12 credits) 7
4. Performance (7 credits) 11

Problem 1 Warming up (9 credits)

a) In the expression `auto x = 64 / 4 - 1 / 2;`, what is the type and value of `x`?

- ☐ `double` and 10.66
- ☐ `int` and 10
- ☒ `int` and 16 → Integer division + division before subtraction
- ☐ `double` and 15.5

b) What is the one best-fitting name-mangled version of the function declaration `const double normalize(double value, int N)` (assuming GCC on x86)?

- ☐ `double normalize(double value, int N) const` → This is not what name mangling means
- ☐ `_cd_normalizedi` → The return type is not part of the function name
- ☒ `_Z9normalizedi` → See slide 3.3.
- ☐ `_Z6normalizei` → This just blindly replicates the example of the slides

c) Given a `std::vector` of capacity 10, with currently 10 elements in it, we add a new element using the `push_back` method. Which one of the following statements best describes what is happening?

- ☐ The vector allocates memory for one additional element next to the old elements and appends the element. All elements are stored next to each other.
- ☒ The vector allocates memory for multiple additional elements, copies the old elements, and appends the element. All elements are stored next to each other.
- ☐ The vector allocates memory for one additional element, copies the old elements, and adds the element. All elements are stored next to each other.
- ☐ The vector allocates memory for additional elements and creates a link to the new location. Not all elements are stored next to each other anymore.

d) Given a `std::unique_ptr<T> up;`, which one of the following statements would lead to a compilation error?

- ☐ `f(up)`, where `void f(std::unique_ptr<T> const & arg);`
- ☒ `f(up)`, where `void f(std::unique_ptr<T> arg);` → a `std::unique_ptr` cannot be copied.
- ☐ `f(std::move(up))`, where `void f(std::unique_ptr<T>&& arg);`
- ☐ `f(up)`, where `void f(std::unique_ptr<T>& arg);`

See also `https://compiler-explorer.com/z/e4fYzbeWP`

e) Which one of the following statements best describes the concept of RAII?

- ☐ Every object that reads arrays via its interface should initialize those arrays.
- ☐ Every object that inherits virtual methods should avoid slicing parent members.
- ☐ Every object that implements an abstract class should implement its interface.
- ☒ Every object that allocates memory should free that memory with its destructor.

f) Which of the following modificiations can usually lead to a significant runtime performance gain? Choose the one most fitting answer.

- ☐ Using `std::vector` instead of a C-style array (i.e., `double a[N];`).
- ☒ Using the GCC compiler option `-O3`
- ☐ Using inline-if instead of a block-if.
- ☐ Using the GCC compiler option `-pg`

0 ☐ 1 ☐ 2 ☐

g) Consider the following code:

```cpp
template<typename T>
set_length(T path, double len){
  path.length = len;
}

int main(){
  double vec = 2.0;
  set_length(vec, 3.5);
}
```

Identify **two** issues in this code: one that will trigger a first-phase lookup error and one that will trigger a second-phase lookup error. **Indicate** which one is first-phase and which one is second-phase

First phase: `set_length` has no return type.
Second phase: `double` is a fundamental type, not a class. As such, it does not have a `length` member.

h) Which one of the following is not initializing `d` to zero?

- ☐ `double d{};`
- ☐ `double d(0);`
- ☐ `double d = 0;`
- ☒ `double d();`

Problem 2 Data types, Functions, STL (8 credits)

It's time to plan your summer vacation! As a good programmer, you want to make a program to help you find the optimal destination.

a) Write a function price_per_day to compute the average price per day, taking into account the cost (floating-point number) of transportation (cost_transportation) and accommodation (cost_stay), as well as the number of days (N_days) (always positive).

0
1

```
float price_per_day(float cost_transportation, float cost_stay, size_t N_days){
  return (cost_transportation + cost_stay) / N_days;
}
```

b) Write a main function, which:

0
1
2
3

1. creates a vector of prices per day for N destinations
2. iterates over the collection (without using an index)
3. reads the costs and number of days from the user input
4. computes the respective price_per_day value for this destination
5. stores this price in the vector

```
int main(){
  size_t N = 10;
  std::vector<float> prices(N);
  float cost_transportation;
  float cost_stay;
  size_t N_days;
  for (auto & price : prices){
      std::cin >> cost_transportation >> cost_stay >> N_days;
      price = price_per_day(cost_transportation, cost_stay, N_days);
  }
}
```

0
1
2

c) Use the C++20 ranges library to convert the costs from Euro (assumed) to AdvProgCoin (exchange rate: 1 Euro = 9AdvProgCoin) and then select and store only destinations with average price per day less than 10 AdvProgCoin.

```
    auto converted = prices
    | std::views::transform( [](auto i){ return i*5;} ) // here 1 Euro = 5 AdvProgCoin
    | std::views::filter( [](auto i){return i<10;} );
```

See the complete code on `https://compiler-explorer.com/z/3s7Yxf35f`

d) When using the algorithm `std::find_if` on an `std::array`, if the element that we are looking for is not found, where does the returned iterator point to?

- ☐ To the found element
- ☒ Next to the last element
- ☐ To `nullptr`
- ☐ To the last element

Problem 3 Object-oriented Programming (12 credits)

Your team is responsible for developing a software for holiday bookings in an object-oriented manner. One core component of such bookings is different destination categories. Figure 3.1 shows the UML class diagram for the intended first design of destinations.

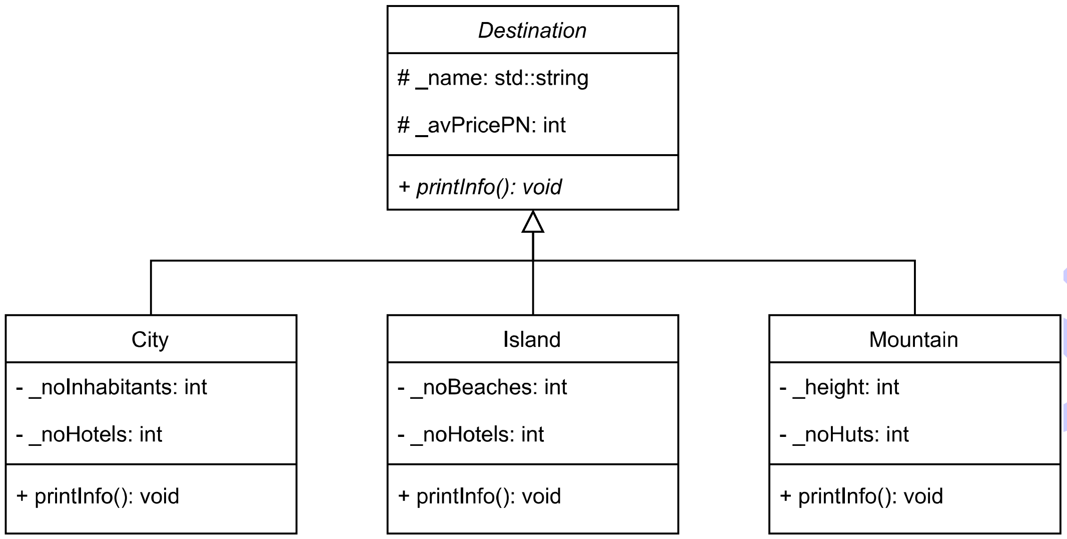

Figure 3.1: UML class diagram showing the design regarding holiday destinations.

a) Implement the base class as sketched in Fig. 3.1. Take care of the following details:
- The member _name in the base class should never change.
- The member _avPricePN in the base class should be initialized with zero.
- The method printInfo() in the base class has to be public and abstract and should never modify an instance of that class.

Both members should only be accessible by this class and by potential subclasses.

0 1 2 3

```
class Destination{
    protected:
        const std::string _name;
        int _avPricePN = 0;
    public:
        Destination(std::string name);
        virtual void printInfo(void) const = 0;
};
```

0
1

b) Indicate the code for the constructor of `Destination` (from task a)), setting the name of the destination from an argument and **as if it was defined in a separate file** (header + implementation).

```
Destination::Destination(std::string name)
: _name(name) {
}
```

0
1
2

c) Implement **only** the child class **`Mountain`** indicated in Fig. 3.1. Take care of the following details:
- The members of the child class need to be private and initialized explicitly via arguments.
- The class has to initialize the base class correctly in the constructor.
- Indicate the necessary declaration for the method `printInfo()`. The definition is not necessary here.

```
class Mountain : public Destination{
    private:
        const int _height;
        int _noHuts;
    public:
        Mountain(std::string name, int height, int noHuts)
        : Destination(name), _height(height), _noHuts(noHuts) {
        }
        void printInfo(void) const override;
};
```

In a next step, the design of destinations shall be extended by a decorating functionality which will allow for the addition of attributes such as "eco-friendly" or "bicycle-friendly". As expected, it should be possible to add these attributes to any of the existing destination classes from above. Figure 3.2 shows the UML class diagram for the design of decorators for destinations.[1]

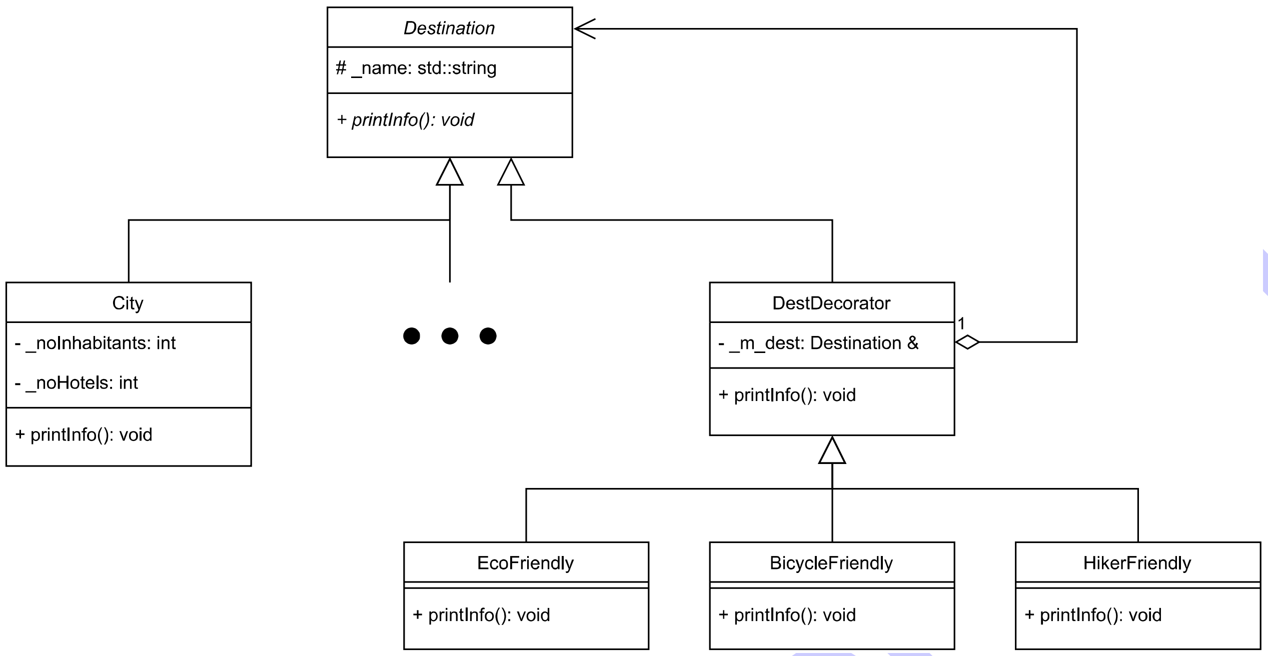

Figure 3.2: UML class diagram showing the extension of destinations by decorator functionality.

d) Implement the class `DestDecorator` as indicated in Fig. 3.2. Take care of the following details:

- The class has a private member `_m_dest` which is a reference to the base class `Destination`.
- The constructor definition sets `_m_dest` in a suitable manner.
- The method `printInfo()` calls the implementation of the corresponding method on the member object.

0 1 2

```
class DestDecorator : public Destination {
    private:
        Destination& _m_dest;
    public:
        DestDecorator(Destination& dest) : _m_dest(dest) {
        }
        void printInfo(void) const override {
            _m_dest.printInfo();
        }
};
```

[1]This hints towards the decorator design pattern, something which we have not discussed, but you already know everything you need to implement it. Think of adding toppings to different kinds of ice cream.

0
1
2

e) Implement **only** the class **EcoFriendly** as indicated in Fig. 3.2. Take care of the following details:
- The constructor obtains a reference to type Destination and hands that over to a call of the constructor of the parent class DestDecorator.
- The implementation of the method printInfo() first calls the variant of the father class DestDecorator and then adds some command line output indicating the type of friendliness.

```
class EcoFriendly : public DestDecorator {
    public:
        EcoFriendly(Destination& dest) : DestDecorator(dest) {
        }
        void printInfo(void) const override {
            DestDecorator::printInfo();
            std::cout << "+␣eco␣friendly" << std::endl;
        }
};
```

0
1

f) Write C++ code to instantiate an object of type Mountain with name "Mont Blanc", with a height of 4806 and with a number of huts of 51. This object should have the variable name myObj. Create a second object which decorates the first by the attribute "eco-friendly" according to Fig. 3.2.

```
    myObj = Mountain("Mont Blanc", 4806, 51)
    EcoFriendly myObj_decorated(myObj);
```

0
1

g) Is it possible to combine different decorating attributes (i.e., make a destination have two "friendliness" characteristics) with the design shown in Fig. 3.2? Give a short argument.

```
    EcoFriendly mad_eco(madeira);
    BicycleFriendly mad_eco_bike(mad_eco);
```

Problem 4 Performance (7 credits)

You are now in the role of a revenue manager and, as a first step, you want to smoothen the prices for different destinations, applying at the same time a uniform price increase.
You want to run the following code on this system:

```
const size_t N = 100000;
std::vector<double> prices(N);

for (auto i=0; i<N; ++i) {
  if ( i == 0) {
    prices[i] = (prices[i] + prices[i+1])/2 + 10.0;
  }
  else if ( i == N-1 ) {
    prices[i] = (prices[i-1] + prices[i])/2 + 10.0;
  }
  else {
    prices[i] = (prices[i-1] + prices[i] + prices[i+1])/3 + 10.0;
  }
}
```

a) What is a potential performance (and readability) issue with this code? Explain why. (assume no compiler optimizations at this point)

0 1

We have branches inside the loop, which could prevent pipelining and vectorization.

b) Simplify the loop to avoid branches.

0 1 2

```
    prices[0] = (prices[0] + prices[1])/2 + 10.0;
    for (auto i=1; i<N-1; ++i) {
        prices[i] = (prices[i-1] + prices[i] + prices[i+1])/3 + 10.0;
    }
    prices[N-1] = (prices[N-2]+prices[N-1])/2 + 10.0;
```

See the code on `https://compiler-explorer.com/z/9vvn7oPYE`

0 ☐
1 ☐
2 ☐
3 ☐

c) You are given a system with peak performance of $P_{\max} = 7\text{GFLOP/s}$ and sustained memory bandwidth $b_s = 88\text{Gbyte/s}$ to run the code from b) on. Is the performance **of the loop** restricted by the computations or by the memory accesses?

- The body of the loop is: `prices[i] = (prices[i-1] + prices[i] + prices[i+1])/3 + 10.0;`
- We have 3 loads (the three prices) of 8 bytes (double) and 1 store (the modified price), i.e., 32 byte.
- We perform 4 floating-point computations: two additions of prices, one division, and one addition. The index computations are not FLOP.
- Computational intensity: $I = 4\text{FLOP}/32\text{byte} = 1/8\text{FLOP/byte}$
- Roofline model: $P = min(P_{\max}, b_s * I) = min(7\text{GFLOP/s}, 11\text{GFLOP/s}) = 7\text{GFLOP/s}$.
- From the roofline model, we see that the code is currently compute-bound.

0 ☐
1 ☐

d) You can upgrade your CPU to one supporting AVX-512, i.e., a CPU that can process 8 vector elements in parallel. Can this feature help you improve the performance? Explain/show. You can assume that the loop length is perfectly divisible by 8.

The code is not vectorizable, as there are loop-carried dependencies (prices[i] is modified in the previous iteration).

Additional space for solutions–clearly mark the (sub)problem your answers are related to and strike out invalid solutions.