Chair of Scientific Computing in Computer Science
Department of Informatics
Technical University of Munich

Eexam
Place student sticker here

Note:

- During the attendance check a sticker containing a unique code will be put on this exam.
- This code contains a unique number that associates this exam with your registration number.
- This number is printed both next to the code and to the signature field in the attendance check list.

Advanced Programming

Exam:	IN1503 / Endterm	**Date:**	Monday 22nd February, 2021
Examiner:	Prof. Dr. Hans-Joachim Bungartz	**Time:**	11:30 – 12:30

Working instructions

- This exam consists of **12 pages** with a total of **3 problems**.
  Please make sure now that you received a complete copy of the exam.
- The total amount of achievable credits in this exam is 36 credits.
- Detaching pages from the exam is prohibited.
- Allowed resources:
  - This is an open-book exam. The exam is designed having in mind that you can look at the **course material** whenever you want, but don't forget to keep an eye on the time!
- Often, subproblems are independently solvable, so make sure to try everything.
- **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.
- Communication with other people during the examination is strictly prohibited.
- 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.

Left room from ________ to ________ / Early submission at ________

Problem 1 Working with `std::vector` (11 credits)

a) What does CMake do? Select only one of the following.

□ It makes a compatibility layer between C and C++.

□ It builds a C++ project.

□ It generates the instructions for a build system.

0 □ □ 1 □ □ 2 □ □ 3 □ □ 4 □

b) Write a C++ function `selectPrint()` that fulfils the following requirements.
The function shall:

- return nothing,
- have a parameter `vec` which represents a `std::vector` over a templatized type `T`, and
- select those entries in `vec` that are larger than zero and print them to the command line.

Do not forget to indicate all necessary include statements.

0 □ □ 1 □ □ 2 □

c) Indicate **two** issues that an incompatible type `T` can cause in the function `selectPrint()` (from b)). Explain.

d) What option do you have in C++ to specify additional restrictions on the type of `T` in `selectPrint()` (from b))? Write one sentence explaining.

□ 0
□
□ 1

e) The following code shall find the overall number of entries of an integer value `target` in the vector of integers `vec`. Indicate **two** runtime/logical errors: point to the corresponding line numbers, give an argument or description what is wrong there, and how each can be fixed.

□ 0
□
□ 1
□
□ 2
□
□ 3

```
1  int & findNumberOfEntries(std::vector<int> &vec, int target) {
2      int numberOfFoundEntries = 0;
3      auto i = vec.begin();
4      while (i != vec.end()) {
5          i = std::find(vec.begin(), vec.end(), target);
6          if (i != vec.end()) {
7              numberOfFoundEntries++;
8              i++;
9          }
10     }
11     return numberOfFoundEntries;
12 }
```

Problem 2 Object-oriented programming (16 credits)

a) Which of the following do we need to achieve runtime polymorphism?
Check all that apply. A wrong "check" removes a point, with the minimum number of points being zero.

□ A sliced object of a derived class

□ Friend classes to allow access to private members

□ A derived object managed through a pointer to a base class

□ Virtual functions

□ A virtual constructor

0 □
□
1 □

b) Consider the following code, which implements a Database class to keep track of people that need an appointment for vaccinations:

```
struct Person{
    std::string name;
    std::size_t age; // age in years
};

class Database{
private:
    Person* _people;
    std::size_t _num_people;
    const std::string _author; // Institute that maintains the database, e.g. "RKI".

public:
    // (nothing here at the moment)
};
```

Write a constructor for Database, which should create a complete and valid state of the object from a given author and num_people and allocates the _people array so that it can store num_people elements. You do not need to give values to the elements of _people.

0 □
□
1 □

c) Write the destructor of Database (from b)) in a way that applies the concept of Resource Acquisition Is Initialization.

d) The institute maintains several databases and often wants to construct a new database as a copy of an old one. Implement the copy constructor of `Database` (from b)).

□ 0
□
□ 1
□
□ 2
□
□ 3

e) While developing `Database`, you quickly realize that you should better rely on existing containers to store your data inside the `Database` class.
Modify the declaration of `_people` (from b)) so that it is a `std::vector` instead of a pointer.

□ 0
□
□ 1

0 □ □ 1 □ □ 2 □

f) Modify the constructor (from b)) and destructor (from c)) accordingly to fit the changes in e).
(Skip the copy constructor – in the following, assume that you also modified the copy constructor here.)

0 □ □ 1 □

g) Implement a getter function `get_people` in `Database` (from e)) that returns the vector of `_people` and can be called from outside the class.

0 □ □ 1 □

h) In `main()`, construct a `new_database` from the `old_database` using only functions/methods defined in the class `Database` (from f)). In case you skipped the previous parts: we are now using vectors and you can assume the copy constructor defined.

```
int main(){
    Database old_database(...); // Assume given

}
```

i) Consider the following algorithm, which rearranges a container based on a condition:

□ 0 □ □ 1 □ □ 2 □ □ 3

```
template< class ForwardIt, class UnaryPredicate >
ForwardIt partition( ForwardIt first, ForwardIt last, UnaryPredicate p )
```

where the predicate is a function that returns `true` or `false` for each element pointed to by the iterators (in the range [*first*, *last*)).

Get the vector of people from `new_database` (from h)) using `get_people`. Using the `std::partition` algorithm, partition the vector into two parts: people with age greater than or equal to 65 and people with age less than 65. The order of the two parts does not matter in this case.

j) Given a **`const`** database, for example:

□ 0 □ □ 1

```
const Database new_database(...);
```

what are the requirements so that the following code compiles? Provide specific code changes, if changes are needed.

```
auto vec = new_database.get_people();
```

Problem 3 Performance analysis, optimization, and vectorization (9 credits)

0 □ □ 1 □ □ 2 □ □ 3 □ □ 4 □

a) Consider the following code kernel:

```
for(auto i = 0; i < points.size(); i++){
    auto elem = points[i];
    result[i] = 8 + elem + 2 * elem * elem;
}
```

where `points` and `result` are of type `std::vector<double>` and of size `N`.
You are ordering a new computer and you have the choice between two processors with the following differences:

Model A: vector units that can perform 8 double-precision FLOP/cycle.

Model B: vector units that can perform 16 double-precision FLOP/cycle.

Both processors have the same frequency (2GHz) and scalar performance (1 FLOP/cycle), while “Model B” is significantly more expensive. In both cases, the memory bandwidth of the system will be 12 GB/s (same for read- and write-operations).
Which processor would you buy, with the only application being the above code kernel? Explain your decision thoroughly using the roofline model analysis.

b) Is the following loop vectorizable? Explain.

□ 0 □ 1

```
// double result[N];
// double arr[N];

for(auto i = 0; i < N-4; i++){
    result[i] += arr[i];
    result[i+1] += arr[i];
    result[i+2] += arr[i];
    result[i+3] += arr[i];
}
```

c) Is the following loop vectorizable? Explain.

□ 0 □ 1

```
// double result[N];
// double arr[N];

for(auto i = 1; i < N; i++){
    result[i] = 2 * arr[i-1];
}
```

d) Consider the following code:

□ 0 □ 1 □ 2

```
//res initialized to zero
// double res[];
// double mat[][];
for(auto col= 0; col < N; col++){
    for(auto row = 0; row < N; row++){
        res[row] +=  mat[row][col] / 2;
    }
}
```

Explain at least two modifications to improve cache efficiency and/or computation costs (assuming no compiler optimizations).

0 □
□
1 □

e) You are developing a simulation program, which needs different digits of pi. The number of needed digits is known at compile time and the `compute_pi` function is (only) called once.

```
double compute_pi(int num_digits){
    // performs expensive computation
}
int main(){
    double pi = compute_pi(8);
    // use throughout program
}
```

Is there any way for you to optimize the runtime behavior of this code? Show any code changes needed.

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