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

Ecorrection
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.

```cpp
#include <vector>
#include <iostream>

template <typename T>
void selectPrint(std::vector<T> &vec) {
        for (const auto& entry: vec) {
                if (entry > 0) { std::cout << entry << std::endl; }
        }
}
```

alternatively with `for_each()`:

```cpp
#include <algorithm>

template <typename T>
void selectPrint_forEach(std::vector<T> &vec) {
        std::for_each(vec.begin(), vec.end(),
                [](T& x){if (x > 0) { std::cout << x << std::endl; }});
}
```

Points:

- 0.5 for all 2 (or 3) include statements
- 1.0 for template correct (template keyword, brackets, typename keyword, T, std::vector<T>), 0.5 for almost correct.
- 0.5 for return type correct
- 0.5 for mentioning a for loop (or `for_each`)
- 0.5 for correct loop conditions (iterators in `for_each`)
- 0.5 for correct if (or lambda)
- 0.5 for correct command line output

Passing `vec` by value is also fine.

0 1 2

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

- A type that does not provide an `operator>()` or is not implicitly convertable to an `int` will make the comparison fail.1 point for any of the two.
- A type for which there is not implementation of `operator<<()` will make the `std::cout <<` fail.1 point

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

`concept` allows to specify constraints on types for templates. E.g. here, we should restrict to datatypes supporting the ">" operator.
Points:
- name `concept` (0.5)
- a valid explanation (0.5)

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 }
```

1. error: lines 1 and 2: local stack variable reference returned (segfault when accessed outside the function) ⇒ use `int` instead of `int &` as return type
2. error: line 6: loop will not terminate due to start of find always at beginning of vector ⇒ use `i` instead of `vec.begin()` as starting iterator

Points:
- correct errors/lines: 2×0.5
- correct explanation of problems: 2×0.5
- correct proposed solutions: 2×0.5

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 This is actually what we are trying to avoid by using pointers.
- ☐ Friend classes to allow access to private members This is not related to runtime polymorphism
- ☒ A derived object managed through a pointer to a base class 1 point
- ☒ Virtual functions 1 point
- ☐ A virtual constructor There is actually no such thing as 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:

```cpp
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`.

```cpp
Database(std::string author, std::size_t num_people)
: _author(author), _num_people(num_people), _people(new Person[_num_people])
{}
```

- 0.5 points for using the member initializer list for `const std::string _author`.
- 0.5 points for initializing all other members correctly.

0 1 c) Write the destructor of `Database` (from b)) in a way that applies the concept of Resource Acquisition Is Initialization.

```cpp
~Database(){
    delete [] _people;
}
```

- 0.5 points for `~Database()` and `delete`.
- 0.5 points for `[]`.

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

```
// Example implementation:
Database(const Database& from) : _author(from._author){
    _num_people = from._num_people;
    _people = new Person[_num_people];
    // copy people from "from"
    for(auto i = 0; i < _num_people; i++ ){
        _people[i] = from._people[i];
    }

    // alternatively, use std::copy
}
```

- 1 point for (const) reference `const Database& from`.
- 0.5 points for using the member initializer list `_author(from._author)`.
- 0.5 points for initializing every other member correctly.
- 1 point for copying existing objects from `from._people`, e.g. using a loop or `std::copy`.
  For almost correct syntax, 0.5 points instead.

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

```
// Instead of: Person* _people;
std::vector<Person> _people;
```

- 1 point for correct member declaration `std::vector<Person> _people;`.
  No points for missing the template arguments `<Person>`.

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.)

```
// Instead of the previous constructor:
Database(std::string author, std::size_t num_people)
: _author(author), _people(std::vector<Person>(num_people))
{
    _people = std::vector<Person>(num_people);
    //_people.reserve(num_people);
}

// Instead of the previous destructor:
~Database() = default;
// ~Database(){};
// no declaration for destructor necessary
```

- 1 point for correct initialization for member variables in constructor, or 0.5 point for `_people` of wrong size e.g. not changing `_people` in constructor
- 1 point for empty `~Database()` or for `=default` or explicitly mentioning that there is no need to declare a destructor.

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.

```
public:
    std::vector<Person> get_people() {
        return _people;
    }
```

1 point for completely correct getter, 0.5 for missing `public`, 0 points otherwise.
Returning by reference is also fine. Returning by const reference is accepted, even though it creates issues later.

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.

```
1  int main(){
2      Database old_database(...); // Assume given
3
4  }
```

```
Database old_database(...); // Assume given
Database new_database(old_database);
// Alternative: Database new_database = old_database;
// Wrong: Database new_database(...);
//        new_database = old_database;
```

1 point for calling the copy constructor to create `new_database` from the given `old_database`.
0 points for using the copy-assignment operator.

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.

```
auto vec = new_database.get_people();
auto mid = std::partition(vec.begin(), vec.end(),
                          [](Person p) { return p.age >= 65;});
// alternative without assigning to mid ok.
// alternative using std::begin: std::partition(std::begin(vec), ...)
// alternative to lambda: using the function bool pred(Person p){...}
```

- 1 point for using the `.begin()` and `.end()` in `std::partition`: `std::partition(vec.begin(), vec.end(), ... );`
- 1 point for the general syntax `[](Person p){}` OR `bool pred(Person p){}`.
- 1 point for correct body of function: `{return p.age >= 65;}` or `{return p.age < 65;}`. 0.5 points for slight mistakes, e.g. `{return p.age > 65;}` or `{return p.age <= 65}` (no ;).

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();
```

The `get_people()` method needs to be `const`-correct:

```
std::vector<Person> get_people() const;
```

1 point for correct function signature `std::vector<Person> get_people() const;`
0 points for vaguely mentioning “const” or putting it in the wrong side.

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

0 1 2 3 4

a) Consider the following code kernel:

```
1  for(auto i = 0; i < points.size(); i++){
2      auto elem = points[i];
3      result[i] = 8 + elem + 2 * elem * elem;
4  }
```

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.

Memory interactions of this code: 1 load + 1 store for each iterations, → 2*N memory interactions.
Total floating point operations: 4*N (2 additions, 2 multiplications).
Computational intensity: (4*N FLOP) / (2*8*N Byte) = 1/4 FLOP/Byte. Remember: one double needs 8 Byte and we have two memory interactions.
Theoretically maximum performance that the code can achieve: 1/4 FLOP/Byte * 12 GByte/s = 3 GFLOP/s.
Peak scalar performance for both systems: 1 FLOP/cycle * 2 Gcycle / s = 2 GFLOP/s (the code would be compute-bound).
The loop can be vectorized. Vector peak performance for "Model A": 8 FLOP/cycle * 2GHz = 16 GFLOP/s. Vector peak performance for "Model B": 16 FLOP/cycle * 2GHz = 32 GFLOP/s.
Since the theoretical maximum performance that the given code can achieve with this memory bandwidth is below the vector performance of both processors (memory-bound), we decide to get the cheaper "Model A".

- 1 point for correct intensity (including units).
  Only 0.5 points for not dividing by 8 Bytes or for forgetting units.
- 1 point for correct peak performance of the code.
- 1 point for correct computation of the peak vector performance of the two models (the scalar performance computation is optional).
- 1 point for showing that it is memory-bound for the vector case and deciding for "Model A".

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];
}
```

No: there are loop-carried dependencies (e.g. in `result[i+1]`). If the loop step was `i+4`, this would not be an issue. 1 point for correct explanation, 0 points otherwise.

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];
}
```

Yes: Straight-line of code with a known number of iterations. 1 point for correct explanation, 0 points otherwise.

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).

- Modification for cache efficiency: swap the order of loops:

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

  Cache lines follow the matrix rows in C++.

- Modification for reducing the computation costs:

```
res[row] +=  mat[row][col] * 0.5;
```

  Multiplication is cheaper than division.

1 point for each correct argumentation. Different optimizations accepted if valid and explained.

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.

```
1       double compute_pi(int num_digits){
2           // performs expensive computation
3       }
4       int main(){
5           double pi = compute_pi(8);
6           // use throughout program
7       }
```

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

Yes, we can define both `compute_pi` and `pi` as **`constexpr`**:

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

- 0.5 points for **`constexpr`** function
- 0.5 points for **`constexpr double`** `pi`

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