Chair of Scientific Computing in Computer Science
School of Computation, Information and Technology
Technical University of Munich

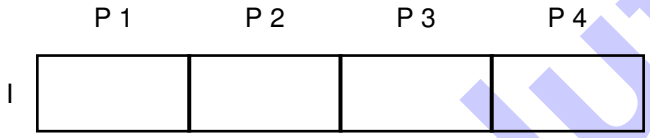
Ecorrection
Place student sticker here

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:**	Wednesday 14th February, 2024
Examiner:	Prof. Dr. Hans-Joachim Bungartz	**Time:**	11:00 – 12:15



Working instructions

- Problems are independent and can be solved in any order. Often, subproblems are also independently solvable, so make sure to try everything.
- Avoid investing too much time on one task. You can gain some points also with partial solutions.
- You do not need to specify required header inclusions or the main function, unless explicitly asked.
- Allowed resources:
  - one sheet of A4 paper (both sides) with hand-written notes (original). No digital, typeset, or printed copies are allowed.
  - one **analog dictionary** English ↔ native language
- Answers are only accepted if the solution approach is documented. Give a reason for each answer unless explicitly stated otherwise in the respective subproblem.
- Physically turn off all electronic devices, put them into your bag and close the bag.
- Do not write with red or green colors nor use pencils.
- There will be an online exam review between February 28 – March 4, during which you can see the solution and how your exam was graded, as well as raise any objections.
- Detaching pages from the exam is prohibited.
- This exam and the sample solution you will receive during the review are property of TUM. Distributing these in any way without written permission is prohibited.
- The total amount of achievable credits in this exam is 42 credits.
- This exam consists of **12 pages** with a total of **4 problems**.
  Please make sure now that you received a complete copy of the exam.

Left room from _____ to _____ / Early submission at _____

Problem 1 Warming up (10 credits)

In all questions, select only one answer. If you want to edit your answer, fill the box of your previous answer completely. This problem is automatically graded, but you can check the grading during the review period.

a) (redacted)

- [x] Function overloading See slide 12.5
- [ ] Structures and enumerations
- [ ] Heap memory allocation
- [ ] A standard library

b) (redacted)

- [ ] Virtual machine
- [x] Assembler See slide 5.6
- [ ] Package manager
- [ ] Interpreter

c) (redacted)

- [ ] `int` and 9
- [ ] `double` and 9.75
- [ ] `double` and 8.0
- [x] `float` and 9.0 See slide 2.15

See https://compiler-explorer.com/z/oscnzdM4v and https://cppinsights.io/s/87479ee9

d) (redacted)

- [ ] `():{}`
- [x] `[](){}` See slide 3.22
- [ ] `:(){ :|:& };:`
- [ ] `void (){}`

e) (redacted)

- [ ] `f(1/2);`
- [ ] `int c = 1; f(std::move{c++});`
- [ ] `f(1<3);`
- [x] `double r = 3.14; f(r);` See slide 4.30

See https://compiler-explorer.com/z/b4jP1963f

f) (redacted)

- ☐ cache lines
- ☒ operations See slide 7.41
- ☐ list elements
- ☐ types

g) (redacted)

- ☐ Its arguments can be enumerated This not possible for templates with neither C++20 nor C++23
- ☐ It describes a variable There is no such thing
- ☒ It uses a parameter pack See slide 8.21
- ☐ It includes a `requires` clause This is a concept

h) (redacted)

- ☐ An error-handling mechanism
- ☒ A view on a contiguous container See slide 12.17
- ☐ A text manipulation STL algorithm
- ☐ A modules-based standard library

i) (redacted)

- ☐ `codenames.begin()->second` In a `std::map`, elements are sorted by key.
- ☒ `(*--codenames.end()).second`
- ☐ `codenames["Silene"].second` The `[key]` will directly give the value
- ☐ `codenames.at("Silene").value` Does not exist

See `https://compiler-explorer.com/z/4P475sWdb`

j) (redacted)

- ☒ No, unless we overload the constructor of `Lightsaber` A new `Lightsaber` object is created
- ☐ Yes, using the default constructor of `Lightsaber` The default constructor takes no arguments
- ☐ No, unless we define the copy constructor of `Blaster` Copy constructor: same type
- ☐ Yes, bitwise, but the `Blaster` object will be sliced off The classes are unrelated, this is not alowed

See `https://compiler-explorer.com/z/j3vME8Kr6`

Problem 2 Data types, functions, STL (12 credits)

(redacted)

0 1

a) (redacted)

```
struct Triplet {
  char    content;
  size_t  word;
  size_t  position_in_word;
};
```

- 0.5pt for `struct`, `Class` with `public:`, `std::tuple<>`, or `std::pair<*,std::pair<*,*>>`
- 0.5pt for `size_t`, `long`, or `uint64_t` (16/32/64). `int` not accepted (signed).
- 0.5pt for `char` (the `std::string` would not work with single quotes)

See the code on https://compiler-explorer.com/z/or389sqb5.

0 1

b) (redacted)

```
#include <vector>
int main() { // Fine if void main(), but technically not correct in standard C++
  std::vector<Triplet> message = ENCODED // Semicolon not needed, but fine (noop)
  return 0; // optional, implied in main()
}
```

- 0.5pt for a `std::vector` with correct syntax. A `std::list` is not compatible.
- 0.5pt for correct initialization. `message(ENCODED)` or `message{ENCODED}` does not work due to the semicolon in `ENCODED` (unintended detail/trap). A ranged `for` loop does not work, because `ENCODED` is not a container.
- 0.5pt for the header

0 1 2 3

c) (redacted)

```
void print(const std::vector<Triplet>& message){
  for (const auto& m : message){     // const or & optional in the loop
    std::cout << m.content << " "; }
  std::cout << std::endl;            // many forgot or misunderstood, we ignored it.
}
```

- 1.5pt for void, const, reference (0.5pt each)
- 1.0pt for correct ranged `for` loop or `std::for_each`. No points for index-based `for` loop.
- 0.5pt for accessing `.content` according to the argument type. For `std::tuple`, `.first`, `[0]`, or `.at(0)` are not accepted, only `std::get<0>(tuple)`. But we did not even cover tuples in the course, so we did not ask/expect
- 0.5pt for printing (`std::cout <<`, `std::endl`)

(redacted)

d) (redacted)

0 1 2

```
std::for_each(std::execution::par,
              message.begin(), message.end(),
              [](auto &a) {
                  if (a.content == '3')
                    a.content = 'E';
                }
              );
```

- 0.5pt for `std::excution::par` (or similar: `std::par`, `std::exec::par`)
- 0.5pt for `message.begin(), message.end()`
- 0.5pt for general lambda syntax: `[](){}`
- 0.5pt for `auto &a` or `Triplet &a`. No points for `const auto &a` or missing `&`.
- 0.5pt for directly editing the element (correctly) instead of calling `return`.

(redacted)

e) (redacted)

0 1 2 3

```
std::stable_sort(message.begin(), message.end(),
                 [](const auto & a, const auto & b){
                   return a.position_in_word < b.position_in_word;
                 }
                );

                   // Second call just differs in the return: (but needs to be stable)
                   return a.word < b.word;
```

- 0.5pt for `std::stable_sort`, at least as the second call (the first one can be `std::sort`).
- 1.0pt for the two arguments. Missing `const` or reference is fine.
- 0.5pt for checking for comparison (order matters, no points for `a > b`)
- 0.5pt for showing the difference to the second call
- 0.5pt for first sorting by `position_in_word` and then by `word`. The reverse does not lead to the correct result, because we are still iterating over the whole message.

Alternative solution with one call:

```
std::sort(message.begin(), message.end(),
          [](const auto& a, const auto& b){
            if(a.word == b.word){
              return a.position_in_word < b.position_in_word;
            } else {
              return a.word < b.word;
            }
          });
```

- 0.5pt for `std::sort` or `std::stable_sort`.
- 1.0pt for the two arguments. Missing `const` or reference is fine.
- 1.0pt for the `if(a.word == b.word)`
- 0.5pt for checking for comparison (order matters, no points for `a > b`)

(redacted)

Problem 3 Object-oriented Programming: Decorator Pattern (11 credits)

(redacted)

0
1

a) (redacted)

```
class Component { // component
public:
  virtual void show() = 0;
  virtual ~Component() = default;
};
```

- 0.5pt for correct class syntax
- 0.5pt for abstract function
- 0.5pt for virtual destructor

See the code on https://compiler-explorer.com/z/TPa7WEMY1.

b) (redacted)

0 1 2

```
class CardText : public Component { // specific component
private:
  std::string _text = "dummy";
public:
  virtual void show() override {
    std::cout << "CardText::show: " << _text << "\n";
  }
};
```

- 0.5pt for correct realisation of inheritance
- 1.0pt for correct default text (private, type, default value)
- 0.5pt for correct `override` (or `virtual`) of `show()`
- 0.5pt for correct semantics of `show()`

c) (redacted)

0 1 2 3

```
class Decorator : public Component {
protected:
  std::shared_ptr<Component> _component;

public:
  Decorator(std::shared_ptr<Component> component) : _component(component) {}

  Decorator() : _component() {} // bonus; used in child classes init. lists
  Decorator(const Decorator&) = delete;          // copy constructor
  }
```

- 0.5 pt for protected
- 0.5 pt for correct constructor with param type
- 0.5 pt for correct assignment of `_component`
- 0.5 pt for correct syntax of delete (or marking as private)
- 1.0 pt for declaration of the two methods (copy constructor, copy-assignement operator) with correct signatures
- bonus: 0.5 pt for default constructor with no or Null pointer

0 1 2

d) (redacted)

```
// fuegt der component Funktionalitaet bzgl. hearts hinzu.
class HeartsDecorator : public Decorator {
private:

public:
  HeartsDecorator(std::shared_ptr<Component> component,
    int amount) : Decorator(component), _amount(amount) {
  }
  virtual void show() override {
    Decorator::show();
    sketchHearts(_amount);
  }
```

- 0.5pt for correct const keyword
- 0.5pt for correct initialization of `_component`: constr. of base class
- 0.5pt for member init list of constructor (due to const) – the order does not matter here
- 0.5pt for correct semantics of `show()` (order, base class access)

0 1 2

e) (redacted)

```
std::shared_ptr<CardText> cardTextPtr = std::make_shared<CardText>();
```

- 0.5pt for correct constructor brackets
- 0.5pt for correct templates
- 0.5pt for `make_shared` correct or correct init of shared pointer
- 0.5pt for correct call of `show()`: ptr dereferencing

Problem 4 **Performance (9 credits)**

(redacted)

a) (redacted)

0
1

The fact that we are using an `std::unordered_map` means that we are most likely constantly having cache-misses. Therefore, the code will tend to be memory bound, i.e., located in region A.

- 0.5pt for correct region (needs to be explicitly stated).
- 0.5pt for correct justification

b) (redacted)

0
1

The `std::vector` is cache-friendly, as all elements are consecutive in memory.

- 0.5pt for correct answer.
- 0.5pt for correct justification

0
1
2

c) (redacted)

We consider as valid both answers that consider and that do not consider SIMD effects. If SIMD was considered, then we expect the correct value in number of elements (16 elements)

SIMD: $\frac{1\ FLOP/item \cdot 16\ items \cdot 2\ Gcycles/s}{4\ cycles} = 8\ GFLOP/s$

Non-SIMD: $\frac{1\ FLOP/item \cdot 2\ Gcycles/s}{4\ cycles} = 0.5\ GFLOP/s$

- 0.5pt for 1 FLOP/item.
- 0.5pt for realizing that it can hold 16 floats (SIMD).
- 0.5pt for considering that division takes 4 cycles.
- 0.5pt for correct answer (1.0pt for non SIMD).

0
1

d) (redacted)

Now we need to do 4 times less cycles per operation, thus the performance should be multiplied by 4. Assuming 20 GFLOP/s of performance with division, one would expect 80 GFLOP/s with multiplication.

- 0.5pt for correct answer.
- 0.5pt for justification.

0
1

e) (redacted)

We only have one memory interaction, as `max_score` is a local variable and thus will be cashed. Also, consider that `scores` is a vector of **`float`**s. Finally, we only consider reading operations, no writing.

$\frac{N\ FLOP}{4 \cdot N\ Byte} = \frac{1}{4} FLOP/Byte$

- 1.0pt for correct answer.

f) (redacted)

0
1

The structure presents holes or gaps between some of its members, which means that we could eventually reduce its size to make it more cache-friendly. To do so, we could declare the member variables ordered in size (largest first)

- 0.5pt for mentioning the presence of holes/gaps in the structure.
- 0.5pt for general description of how to fix the issue (i.e., re-order the variables).

g) (redacted)

0
1

It occupies 2 cachelines. This is clearly stated with the comment that indicates the boundary of the first cacheline.
Another possible justification is to compare the size of the structure (80 bytes: 71 B for members + 8 B for holes), with the size of one cacheline as reported by pahole (64 Bytes).

- 0.5pt for correct answer.
- 0.5pt for justification.

h) (redacted)

0
1

Since the vector spans from position 40 to 64, we can establish that occupies 24 Bytes in memory.

- 0.5pt for correct answer.
- 0.5pt for justification.

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