Solution advpr endterm

Advanced Programming (IN1503) (Technische Universität München)

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:**	Monday 13th February, 2023
Examiner:	Prof. Hans-Joachim Bungartz	**Time:**	13:00 – 14:15

	P 1	P 2	P 3	P 4
I				

Working instructions

- 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.
- Allowed resources:
  - one **hand-written sheet of A4 paper (both sides)** with notes
  - 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.
- This exam consists of **16 pages** with a total of **4 problems**.
  Please make sure now that you received a complete copy of the exam.
- The total amount of achievable credits in this exam is 42 credits.
- Detaching pages from the exam is prohibited.
- Do not write with red or green colors nor use pencils.
- Physically turn off all electronic devices, put them into your bag and close the bag.

Left room from __________	to __________	/	Early submission at __________

Problems

Problem 1 Warming up (8 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)

- [✓] `int` and 0 → Integer division 1/2
- [ ] `double` and 0.66
- [ ] `char` and 1
- [ ] `float` and 0.5f

See `https://compiler-explorer.com/z/rzTTjhd8Y`

b)

- [✓] `g++ main.cpp -o main.exe && ./main.exe` → Option `-o` for output, name does not matter on Linux
- [ ] `gcc main -e main.exe && ./main.exe`
- [ ] `gcc main.cpp && ./main` → No output file name is defined, the default is `a.out`
- [ ] `g++ -c main.cpp && ./main.o`

c)

- [ ] On the heap
- [ ] On the free store
- [ ] On the reference
- [✓] On the stack → RAII: They destroy the object they point to when out-of-scope (stack unrolled)

d)

- [ ] the result of `std::make_unique<int>(a)`, where `int a = 0`
- [ ] the literal `3.14`
- [ ] the result of `std::move(b)`, where `double b = 1.23`
- [✓] the variable `x`, where `int x = 2`. → passing an lvalue to an rvalue reference

See `https://compiler-explorer.com/z/scfTzz6Kx`

e)

- [ ] `std::unique_ptr<Fruit> fruit = std::make_unique<Fruit>();`
- [ ] `std::unique_ptr<Apple> fruit = std::make_unique<Apple>();`
- [ ] `auto fruit = std::make_unique<Fruit>();`
- [✓] `std::shared_ptr<Fruit> fruit = std::make_shared<Apple>();` → pointer to base class

See `https://compiler-explorer.com/z/s47Wx4fbE` f)

- ☐ The code does not compile → only the pointer is `const`, not what it points to
- ☒ The code compiles and results to `f(*b) = 43` and `a = 42` → pass-by-value, so `a` does not change
- ☐ The code compiles and results to `f(*b) = 42` and `a = 42`
- ☐ The code compiles and results to `f(*b) = 43` and `a = 43`

See `https://compiler-explorer.com/z/WT8cf7zfn`

g)

- ☐ Move constructor
- ☐ Copy assignment operator
- ☒ Copy constructor → The object `a2` does not exist yet
- ☐ Move assignment operator

See `https://compiler-explorer.com/z/M1fGojrPG`

h)

- ☒ `Pancake pancakeChoco; eat(pancakeChoco);` → Cannot call `Pancake::eat()` without arguments.
- ☐ `Berry pumkin; eat(pumkin);`
- ☐ `Chocolate choco; eat(choco);`
- ☐ `const Soup noodleSoup; eat(noodleSoup);`

See `https://compiler-explorer.com/z/r1dvxT78j`

Problem 2 Data types, functions, STL (11 credits)

a)

0
1
2

```
#include <iostream>

int main(){
  double priceCap = 0.20;
  std::cout << "Price cap = " << priceCap << std::endl;
  return 0;
}
```

- 0.5pt for `priceCap` type and initialization (`float`, `auto`, `const` is also fine)
- 0.5pt for `std::cout << priceCap` (additional text does not matter)
- 0.5pt for `<< std::endl` or `<< "\n "`
- 0.5pt for `return 0;` or `return EXIT_SUCCESS;` No points for `return 1;` or `return true;` This is implied for `main()`, did not remove points if omitting was the only issue.
- 0.5pt for including `<iostream>`
- -0.5pt for missing `main(){}` or for having wrong argument types

b)

0
1

```
std::vector<std::pair<std::string, double>> providers;
```

- 0.5pt for `std::pair`
- 0.5pt for `<std::string, double>`
- 0.5pt for `std::vector`
- -0.5pt for not writing `std::` wherever needed
- Alternatively, 0.5pt for correct `std::map` (but no full points, as it is not contiguous)

c)

0
1

```
providers.push_back({"A", 0.15});
providers.push_back({"B", 0.42});
providers.push_back({"C", 0.14});
// Alternatively:
// providers.push_back(std::pair<std::string, double>("A", 0.15));
// See more on https://compiler-explorer.com/z/e96d1G8YK
```

- 0.5pt for `.push_back()`
- 0.5pt for providing pairs (or correct map elements, if map defined in b)
- -0.5pt for wrong kind of braces
- OK if only missing quotes or using single quotes, but -0.5pt if multiple issues

0 ☐
1 ☐
2 ☐

d)

```
for (auto & provider : providers){
  if (provider.second > priceCap){
    provider.second = priceCap;
  }
}
```

- 0.5pt for ranged-for loop syntax (no points for classical)
- 0.5pt for `auto &` or `std::pair<std::string, double> &` (no points for `const` or pass-by-value)
- 0.5pt for `provider.second`
- 0.5pt for `if` syntax
- Correctly using structured bindings to directly capture the price is also fine:
  `for (auto& [_, price] price : providers)`
- -0.5pt for writing `provider[1]` for pairs, or for accessing the map (from b) in a wrong way.
- Some students misunderstood the “reduce” as a reduction operation (sum) of all elements and storing them into `priceCap`. No points lost if the implementation was correct. -0.5pt for setting all elements to or `priceCap` or for subtracting `priceCap` from all elements.

0 ☐
1 ☐
2 ☐
3 ☐

e)

(Algorithm description: excerpt from `https://en.cppreference.com/w/cpp/algorithm/sort`)

```
std::sort(providers.begin(), providers.end(),
          [](auto & a, auto & b) {return a.second < b.second;});
```

- 1.0pt for `std::sort(providers.begin(), providers.end(), )`
- 0.5pt for lambda syntax: `[](){}` (or normal function with correct syntax)
- 0.5pt for returning something boolean
- 0.5pt for arguments list (correct types; references and const do not matter)
- 0.5pt for `a.second < b.second` (with correct direction, only for pairs)
- See various approaches at `https://compiler-explorer.com/z/e96d1G8YK`

0 ☐
1 ☐

f)

```
std::cout << "The cheapest provider is: " << providers.at(0).first << std::endl;
```

- 0.5pt for `.at(0)`, no points for `[0]`
- 0.5pt for `.first`
- Half points for a correct implementation with `throw`

Problem 3 Object-oriented programming (13 credits)

a)

```
class PowerPlant{
    protected:
        const double _maxPower;
    public:
        PowerPlant(double maxPower);
};
```

- 0.5pt for `protected`
- 0.5pt for `const`
- 0.5pt for correct class declaration. Missing semicolon after the class is fine.
- 1.0pt for constructor (0.5 existence & public, 0.5 syntax & parameter (type))

0 1 2

b)

```
        virtual double computeExpectedPower() const = 0;
```

- 0.5pt for syntax (no argument, virtual, abstract, correct return type).
- 0.5pt for const.

0 1

c)

No, because the abstract method `computeExpectedPower()` makes the class abstract.

- 0.5pt for correct answer.
- 0.5pt for argument.
- No argument points for just "virtual". It needs to be "pure virtual" or "abstract".

0 1

0 ☐
1 ☐
2 ☐

d)

```
class WindTurbine : public PowerPlant {
    private:
        const double _rho = 1.225;
        const double _rotorDiam;
    public:
        WindTurbine(double maxPower, double rotorDiam);

        double computeExpectedPower() const override;
};
```

- 0.5pt for constructor with 1 additional param (+ the maxPower)
- 0.5pt for 2 member variables of correct type + const
- 0.5pt for same parameter (type) + return type as in base class
- 0.5pt for const method
- in-task bonus of 0.5pt (same max score) if using `override`
- `_rho` can also be initialized in the constructor (task e))

0 ☐
1 ☐
2 ☐

e)

```
WindTurbine::WindTurbine(double maxPower, double rotorDiam) :
  PowerPlant(maxPower), _rotorDiam(rotorDiam) {
}
```

- 0.5pt for the separate file and namespace `WindTurbine::`
- 0.5pt for (correct) member initialisation list (necessary due to const and base class const)
- 1pt for initializing a `PowerPlant` object. Note that we cannot initialize each member separately (without calling `PowerPlant(maxPower)`) because there is no default constructor for `PowerPlant`.
- If `_rho` does not have default value in declaration (in task d)), then it must be initialised here (otherwise 0.5 subtracted).

f)

0
1

```cpp
double WindTurbine::computeExpectedPower() const {
    return _maxPower * 0.6;
}
```

- 0.5pt for syntax consistent with task d)
- 0.5pt for correct semantics (factor 0.6 etc.)

g)

0
1
2

```cpp
double WindTurbine::computeCurrentPower(double windSpeed) const {
    double power = 0.125 * _rho * _rotorDiam*_rotorDiam * M_PI
                 * windSpeed*windSpeed*windSpeed;
    return std::min(power,_maxPower);
}
```

- 0.5pt for correct input + output (and type)
- 1pt for correct implementation of the formula: access to `_rho` and `_rotorDiam` (radius!)
- 0.5pt for correct power computation of v (no points for `windSpeed^3`)
- 0.5pt for `std::min` or similar to limit the max power
- in-task bonus of 0.5pt for some include of header for π such as cmath or hard-coded approximated value (same max score in this problem)

h)

0
1

The method `computeExpectedPower()` can be a constexpr since everything is known at compile time

- 0.5pt for correct method
- 0.5pt for a valid argument

Problem 4 Performance (10 credits)

0 1 2 3

a)

- Each iteration performs 2 FLOP (one multiplication and one addition).
  We have $NB * NM$ iterations.
  Thus, we have total **operations:** $2 * NB * NM$ FLOP.
- Data from `consumptions` is loaded $NB * NM$ times.
  Data from `price_list` is loaded NB times.
  `total_cost` is a local variable, so it does not require any memory interactions.
  In total, we have $NB * (NM + 1)$ **memory interactions** (all 4 Byte floats, all reads).
- The computational intensity is:

$$I = \frac{operations}{memory\ interactions * 4\text{Byte}} = \frac{NM}{2 * (1 + NM)} \frac{\text{FLOP}}{\text{Byte}} \tag{4.1}$$

- 1.0pt for correct operations.
- 1.0pt for correct number of memory interactions (0.5pt for each `std::array`).
- 1.0pt for correct calculation of computational intensity and units
  (if wrong because either FLOP or number of memory interactions are wrong, consider the correct use of the equation, including the division by 4).

b)

0
1

The floating point performance will be given by:

$$P = min(I * b_s, Peak) = min(0.5 * 1, 2) = 0.5 \frac{\text{GFLOP}}{\text{s}} \tag{4.2}$$

- 0.5pt for correctly stating the formula.
- 0.5pt for the correct value.

c)

0
1

Yes. We traverse the code in a row-major order.
- 0.5pt for the Yes
- 0.5pt for the correct justification.

d)

0
1

No. The `price_list` object is of type `std::list`, which is not guaranteed to be contiguous in memory.
- 0.5pt for the No
- 0.5pt for the correct justification.

0
1
2
3

e)

```
std::array<float, NB> price_list;
std::array<std::array<float, NM>, NB> consumptions;

// Initialization of price_list, consumptions ...

float cost = 0;
for (int i = 0; i < consumptions.size(); i++) {
  for (int j = 0; j < K; j++)
    cost += consumptions[i][j] * price_list.at(i) * (80.0 / 100.0);
  for (int j = K; j < consumptions[i].size(); j++)
    cost += consumptions[i][j] * price_list.at(i);
}
```

- Change from `std::list` to `std::array` (`std::vector` also valid).
- Replace the if-else statements with two for loops.
- 1.5pt per optimization (0.5pt for valid optimizations + 1.0pt for including the code)

0 ☐
1 ☐

f)

No. 80.0/100.0 is a float constant expression which will be calculated at compile time.
- 0.5pt for the No
- 0.5pt for the correct justification.

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