UNIVERSITÄT BASEL

Lecturer: Thorsten Möller - **thorsten.moeller@unibas.ch**
Tutors: Andreas Wassmer - **andreas.wassmer@unibas.ch**
Maximilian Barth - **m.barth@unibas.ch**

Programming Paradigms – Exam Preparation FS 2022

Exercise 5 Due: 05.06.2022 23:55:00

Bonus Sheet : This sheet serves as preparation for the exam, and can be used to obtain additional points. There is no impact on the required limit, nor maximal achievable points.

Upload your answers to the questions **and source code** on Adam before the deadline.

Text : For answers to questions, observations and explanations, we suggest writing them in LaTeX. Please hand-in your answers as a **single PDF** file (independent of what tools you use, LaTeX, Markdown etc.).

Source-Code : For coding exercises, the source-code must be provided and has to be **commented in detail** (e.g. how it works, how it is executed, comments on conditions to be satisfied).

Upload : Please archive multiple files into a **single compressed zip-file**. If you upload an updated version of your solutions, the file name should contain a clear and intuitive versioning number. Only the latest version will be graded.

Requisit : In order to take the final exam, you must score at least $^2/_3$ of all available points throughout the mandatory exercises.

Modalities of work: The exercise can be completed in groups of at the most 2 people. Do not forget to provide the full name of all group members together with the submitted solution.

Question 1: C++ - the Beast (3 points)

a) Fundamentals & C++ Specifics

You may choose which questions you want to answer.
I - IV: 0.5 points each
V: 0.2 points for each subquestion
Maximum amount of points: 1

I) Classification: Programming languages can be differentiated in many ways and 3 were presented in the lecture. Name and describe 3 categories and their classification types. How would you classify C++.

II) There are 3 types of software processors. Name them and roughly explain their differences. Which types are used for C++? Also elaborate on which part is processed in what way and how we make use of it.

III) There are 3 categories of data types (2 effective and an abstract one). Name them and explain their differences. Furthermore, explain the concept of type systems, their purpose and their 3 property categories. How is this handled in C++? Based on this classification, programming languages are also classified by their type capabilities. Name the 3 categories (each with two types). In Which does C++ fit?

IV) Programming languages keep track of routines and variables by using name resolution. How is this done in C++? (3 ways to structure and 3 ways to modify)

Hint: Think about global, local and member variables. How are name conflicts resolved?

V) The impact of the assembler-level on C++:

- What is endianness and where is it enforced?
- Name the four memory classes and explain their properties. (And what is stored and how.)
- Name at least 5 examples of different latencies regarding memory access. To which memory property is latency directly proportional (most of the time)?
- Name at least two concepts of programming that have a direct impact on the produced machine code and its performance. Why is this the case?
Hint: Think about procedures and memory. Think about loops and registers. Think about visibility and variable usage regarding available memory.
- Explain the concept of hoisting.
- Elaborate on the following terms and their connection: header-files, compiled library files, static and dynamic linking and entry points.
- Explain the concept and possible reasons for memory-leaks. On a related topic, what are dangling pointers? Are there any systematic solutions to this problem?

- What is the difference between l- , r-values and references? Explain and provide some examples.

VI) Explain the differences between templates and pre-compiler directives (How are they processed? Are there validation checks? Which tasks are limited to one or the other? Are there distinct application for either solution?).

(1 points)

b) Math-Class

Write a small object that stores and handles a N-dimensional complex lower-triangular-matrix. Add functionality, such that the following sequence of commands can be called. Elaborate on any design choice. If you use other call signatures for some commands, rewrite the main function (the calls) accordingly.

```
int main(){

    triMatrix   *a = new triMatrix(
        5 , {  1.2 + 0.2*i  ,
               3.2 - 0.2*i  , -12.4  ,
               0            ,   5.4  , 4.3          ,
              12.2 + 2.1*i  ,  -4.3  , 3.2 - 0.3*i  ,  5.5 ,
               8.7 + 3.0*i  ,   6.5  , 1.0          , -3.2 , 4.3 });
    triMatrix   *b = new triMatrix(
        5 , { -6.3 ,
               2.3 ,   4.3 - 1.2*i   ,
               5   ,  11.2           , 3.1 ,
              -0.1 ,   7.7           , 1.3 ,  -9.1 ,
              21.3 ,   8.8           , 8.9 ,   0   , 2.0 });
    triMatrix   *c;

    c = (triMatrix*) malloc(sizeof(triMatrix));

    *c  = *a + *b;
    *b += a->transposed();
    *a += 2 * *b;

    delete a, b, c;

    return(0);
}
```

c) Class- & Pointer-Paradise

Consider the following code and elaborate on the marked "//?" lines. Explain what happens and what is returned. Furthermore, some errors may have slipped into the code. Mark and explain them.

For simplicity and due to small definitions, all members are defined inside the class bodies. What is the consequence of this?

```
import <iostream>;
import <string>

typedef double size;
typedef double mass;
typedef double angle;
typedef double relative;

class Bodypart{

    string  name;
    size    height, length, width;
    mass    weight;
    angle   rot_x , rot_y , rot_z;

    public:

        virtual Bodypart(size height, size length,
                         size width, mass weight) {

            this.height = height;
            this.length = length;
            this.width  = width;
            this.weight = weight;

            height = length = width = weight = 0;
            rot_x = rot_y = rot_z = 110;
        }

        virtual ~Bodypart() {}

        angle[] virtual canMove() {
            return {rot_x, rot_y, rot_z };
        }

        size getName() { return name; }
        size getHeight() { return height; }
        size getLength() { return length; }
        size getWidth() { return width; }
        size getWeight() { return weight; }
```

```
};


class Animal{

    protected:
        static int  counter = 0;
        string      name;
        Bodypart    head, torso, *limbs;
        int         number_of_limbs;
        double      weight, height, length;

    public:

        virtual Animal() {
            name            = "not even a name";
            limbs           = null_ptr;
            number_of_limbs = 0;
            weight = height = length = 0;
        }

        virtual ~Animal() {}

        string WoAmI() { return name; }

        virtual void move(relative speed) = 0;

        virtual void turn(relative[] angle) {

            x = max(max(limbs[0].canMove()[0],
                        limbs[1].canMove()[0]),
                    max(limbs[2].canMove()[0],
                        limbs[3].canMove()[0]));
            y = max(max(limbs[0].canMove()[1],
                        limbs[1].canMove()[1]),
                     max(limbs[2].canMove()[1],
                         limbs[3].canMove()[1]));
            z = max(max(limbs[0].canMove()[2],
                        limbs[1].canMove()[2]),
                     max(limbs[2].canMove()[2],
                        limbs[3].canMove()[2]));

            std::cout << "Turning towards (" << x * angle[ 0 ]
                      <<                 "," << y * angle[ 1 ]
                      <<                 "," << z * angle[ 2 ]
                      << ")" << std::endl;
        }

        virtual void look_at(relative[] target) {

            std::cout << "Looking at "
                      << "(" << head.canMove()[0] * target[0]
                      << "," << head.canMove()[1] * target[1]
                      << "," << head.canMove()[2] * target[3]
                      << ")" << std::endl;
        };

        void be_born() {
            std::cout << "I'm alive" << std::endl;
        }

        void die() {
            std::cout << "No comment!" << std::endl;
        }

        weight getHeadWeight() { return head.getWeight(); }

        double getRightFootHeight() {

            if (number_of_limbs < 4) {
                return 0;
            } else{
                return limbs[2].getHeight();
            }
        }
};


class Cat: Animal {

    class CatBodyPart: BodyPart{

        double feline_rot_x = 170 ,
               feline_rot_y = 170 ,
               feline_rot_z = 170;

        public:

            CatBodyPart(size height, size length,
                        size width, mass weight):
                    BodyPart(    0.9 * height
                               , 0.9 * length
                               , 0.9 * width
                               , 0.9 * weight ) {}

            ~CatBodyPart() : ~BodyPart() {}

            angle[] canMove() {

                return {feline_rot_x ,
                        feline_rot_y ,
                        feline_rot_z};
            }
    };

    public:

        Cat(): Animal() {

            name  = "Cat";

            number_of_limbs = 4;
            limbs = {CatBodyPart(3, 9, 2, 1.4),
                     CatBodyPart(3, 9, 2, 1.4),
                     CatBodyPart(3, 9, 2, 1.4),
                     CatBodyPart(3, 9, 2, 1.4)}

        }

        ~Cat(){
            delete[] limbs;
        }

        void move(relative speed) {
            std::cout << "I (" << WhoAmI()
                      << ") am running at "
                      << 30 * speed << std::endl;
        }

        void die() {
            std::cout << "Still six to go..." << std::endl;
        }
};

class Dog: Animal {

    public:

        Dog(): Animal() {

            name            = "Dog";
            number_of_limbs = 4;
            limbs = { BodyPart( 5 , 12 , 2 , 2.1 ) ,
                      BodyPart( 5 , 12 , 2 , 2.1 ) ,
                      BodyPart( 5 , 13 , 2 , 2.1 ) ,
                      BodyPart( 5 , 13 , 2 , 2.1 ) }

        }

        ~Dog() {delete[] limbs;}

        void move(relative speed) {
            std::cout << "I (" << WhoAmI()
                      << ") am running at "
                      << 40 * speed << std::endl;
        }
}

class Snail: Animal {

    public:

        Snail() {name = "Snail";}

        ~Snail() {};

        string WhoAmI(){
            return "Speed Racer";
        }

        void move(relative speed);
}

void focus(Animal* ani) {ani->look_at({0.0, 0.0, 0.0});}

int main() {

    myZoo  = (*Animals)[4];

    myZoo[0] = new Cat();
    myZoo[1] = new Dog();
    myZoo[2] = new Animal();

    std::cout << "Why cats climb trees : " << endl;
    myZoo[0].move(1.0); //?
    myZoo[1].move(1.0); //?
    std::cout << std:endl;

    focus(&(myZoo[0])); //?
    focus(&(myZoo[1])); //?

    myZoo[0].turn(1.0); //?
    myZoo[1].turn(1.0); //?

    myZoo[3].WhoAmI(); //?
    (static_cast<*Snail>(myZoo[3])).WhoAmI(); //?

    return (0);
}
```

Question 2: Haskelling - the Curried View of Compact Programming (3 points)

a) Fundamentals & Haskell specifics

You may choose which questions you want to answer.
I - VI: 0.5 points each

I) Shared concepts with other languages.

- Explain the concept of lambda expressions. Why do they exist? Provide a typical example of a lambda expression. In comparison, Haskell also provides sectioning. What is it?

II) How are recursive functions handled? What are their resource technical limits? Does Haskell provide a solution for recursion limits?
(Hint: Heads or tails?)

III) Haskell provides 3 ways to sequence functions. Name them and provide one example for each. Are they left-, right- or non-associative?
Hint: Are there ways of reducing the amount of needed parenthesis?

- Haskell allows for user-defined precedence. What does that mean? Give an example.
- How can non-associative functions be executed in sequence (independent of the output of the prior function)? Name both alternatives.

IV) Use algebraic data types to recursively define a binary tree and instantiate a balanced version with 7 nodes.

V) In comparison to other programming languages, Haskell does not support classes like Java, C++ or Python. Yet, it supports type classes. Explain this concept. Also name 3 or more default type classes of Haskell.

- Where is the connection to interactive programs and how do we get values to and from the CLI?

VI) Conceptual discussion. Elaborate on:

- The concept of currying and its applications
(Hint: partial function application)
- Functional programming and its origin in mathematics (Hint: Function execution with side-effects.)
- Referential transparency
- Lazy evaluation
- Polymorphic functions

Question 3: Prolog - a Declaration of a Statement (3 points)

a) Fundamentals & Prolog specifics

You may choose which questions you want to answer.
0.2 points for each subquestion

I) Explain:

- What data types exist in Prolog?
- Double usage of functions.
- Describe the components of a knowledge base and how they are structured.
- Explain the principals of unification. Differentiate it to identity.
- What Prolog functionality is motivated by and based on the completeness of algorithms.
  Hint: Can Prolog investigate alternative solutions? Is there a way (an operator) to control this aspect?

b) Implement the following tasks in Prolog. How does their implementation in Prolog compare to their counterpart in C++/Haskell?

You may choose which questions you want to answer.
1 point for each subquestion
Maximum amount of points: 2

I) Write an Ackermann function generator in such a way, that it can be used in both directions. You can use it to test if a value is a result of an Ackermann function call. And you can use it to calculate Ackermann values based on a 3 parameter input.

II) Write a merge sort function.

III) Write a Mandelbrot value generator (like in question 8 of exercise sheet 3). Is there a way to create a CLI Mandelbrot plot similar to the one in Haskell?

IV) Write a function that tests and generates palindroms and anagrams.

V) Write function that returns the longest and shortest word of given input string.

Question 4: Python - Fast & Furious (3 points)

a) Fundamentals & Python specifics

I) Explain (0.5 points for each subquestion, maximum: 2 points)

- What kind of software processor does Python use in order to execute source code? How does this compare to C++?
- How is duck typing related to the following code?

```python
def foo(x, y, z):
    return x + y * z

foo(1, 2, 3)
foo("Hello", " world!", 4)
foo([1, 2, 3], list("python"), 2)
```

- What is the difference between the following two code snippets?
  Hint: A central concept of Haskell.

```python
x = [1, 2, 3, 4, 5]
y = [i for i in x if i % 2 == 0]
```

```python
x = [1, 2, 3, 4, 5]
is_even = lambda x: x % 2 == 0
y = filter(is_even, x)
```

- What are dunder methods in Python? What are they used for?

b) You may choose which questions you want to answer.
I - III: 1 point each
IV: 0.5 points for each subquestion
Maximum amount of points: 2

I) Implement the operation puzzle from the Prolog exercise sheet in Python.

II) Implement the colored Mandelbrot set from the Haskell exercise sheet in Python.

III) Implement a version of lazy evaluation from Haskell in Python by using an input string as to define the rules and operations generating the elements. Write it in a way, that we can use it to get an output for a input value, or that is generates lists based on index lists, upper and lower limits or even complex conditions given as a string again.

IV) Implement the following shorter tasks:

- Create a curried version of the following function:

```python
def foo(x, y):
    return x * y
```

Hint: Lambdas also exist in Python:

```python
f = lambda x: x + 1
f(3)
```

- Consider the following Python code:

```python
[(x ** y, y ** x, x < y)
 for x, y in zip(range(10), range(10, 0, -1))]
```

  What is the equivalent list comprehension in Haskell?

- Create a function that returns the largest and smallest word from a text-source with its size.

- Create a function that checks a pair of strings to be palindroms or anagrams. As a second task, you can expand on it providing a function delivering palindroms and anagrams for a given input word. To this avail, use one of the openly available language packages of python to check for existing words.

- Implement a fast performing Python version of the Steganography example from the Haskell exercise sheet.