UNIVERSITÄT BASEL

Lecturer: Thorsten Möller - **thorsten.moeller@unibas.ch**
Tutors: Nafia Nusrat - **nafia.nusrat@unibas.ch**
Sexhi Picaku - **s.picaku@stud.unibas.ch**
Mark Starzynski - **mark.starzynski@unibas.ch**

Programming Paradigms – C++ FS 2025

Exercise 2 Due: 04.05.2025 23:55:00

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**, preferrably just the files without any folder hierarchy. The naming convention for your zip file is **cs109-e<ExNr>-<Name1>-<Name2>.zip**. 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: Set Implementation (7 points)

Implement an ordered `Set` class for characters; i.e., for each pair of characters c_1, c_2 contained in the set, $c_1 \neq c_2$ and a total ordering is provided on its elements. The elements are ordered based on the ASCII encoding (see https://en.wikipedia.org/wiki/ASCII), defined by a sort function provided by the `Set` class.

You are free to chose an underlying data structure as the basis of your implementation. In the simplest case it could be based on an array-structure extended by uniqueness checks on insertion. Alternatively, you might want to use a binary search tree (BST) as the underlying data structure to get a better performing implementation, which would also allow to implement a dynamic set whose capacity is not fixed rather easily.

All in all, your implementation must provide the following functions at least. Finally, keep memory management in mind.

```

/* An insert function. The function returns 1 if the
character was added, and 0 if it was already in the set. */
int add(int i);

/* A removal function. The function returns 1 if the
character was removed and 0 if it was not in the set. */
int remove(int i);

/* A test function. The function returns 1 if the set contains
the character and 0 otherwise. */
int contains(int i);

/* Computes the current size of the set. */
unsigned int size();

/* A function that outputs all characters in the set on the
console, delimited by commas. */
void print();

/* A function that releases allocated memory. Make sure,
that after this function call, allocated memory was
correctly deleted. */
void destroy();
```

Question 2: Priority Queue, Binary Heap (10 points)

You will implement a simple priority queue whose elements are strings and integer priorities based on a binary max heap. If you are unfamiliar with binary heaps or priority queues, you can read more at:

- https://en.wikipedia.org/wiki/Priority_queue
- https://en.wikipedia.org/wiki/Binary_heap

a) Write a class `PriorityQueueElement` (with a header file) that implements a single element of the binary-heap-based priority queue.

   1. Each element should contain:
      - A **string** (the element's value).
      - An **integer priority** (higher values mean higher priority).
   2. Provide the following:
      - A constructor that takes only the string value of an element as an argument (assume some default priority).
      - A constructor that takes the string value and an integer priority.
   3. Each `PriorityQueueElement` should have:
      - A getter for the string value.
      - A getter for the integer priority.
      - Optionally, a setter for the priority if needed (not strictly required, but may be helpful).

(4 points)

b) Write a class `PriorityQueue` (with its own header file) that uses an **binary heap** to store the values. Implement **max-heap behavior** by priority: the element with the highest priority must be at the top of the heap. Implement at least the following methods:

   1. Constructor and destructor.
   2. A method `put` that inserts a string with a given priority into the priority queue.
      - This should place the new element in the heap and *re-heapify* (bubble up) if needed so that the max-heap property remains valid.
   3. A method `pop` that returns the string with the **highest priority** in the queue and *removes* that element from the heap.
      - This should remove the top element of the binary heap (the highest priority), then *re-heapify* (bubble down) the remaining elements.

4. A method `checkCapacity` that ensures the heap has enough capacity. If not, it doubles it. Optionally, you may also implement capacity reduction if the heap gets very empty.
5. Further methods to ensure the heaps properties (max-heapify).

Note:

Ensure that after each `put` and `pop`, the element with the highest priority always resides at the front (top) of the heap.

(4 points)

c) Write a main method (in a separate `main.cpp`) to test your implementation. Your tests should showcase the full functionality of your implementation.

1. Insert at least **5 different strings** with different integer priorities.
2. Repeatedly call `pop` to retrieve and print each highest-priority string until the queue is empty.
3. Verify that the output is in the order of **descending priority**.
4. Check if the heap capacity is adjusted dynamically.

To test all the functionality of your priority queue you can use the following data:

```
pq.put("LowPriorityTask", 1);
pq.put("MediumPriorityTask", 5);
pq.put("CriticalBugFix", 10);
pq.put("ImportantEmail", 7);
pq.put("VeryImportantEmail", 8);
pq.put("RoutineCheck", 2);
pq.put("ExtinctionEvent", 99);
```

(2 points)

Question 3: Classes and Inheritance (9 points)

In this task you will implement a simple property management application.

a) Define an abstract class `Employee` with the following private fields:

- `String name`
- `String position`
- `double salary`

Provide a constructor to initialize these fields, along with getter and setter methods for each. Implement an abstract method `calculateBonus()` which will be implemented by subclasses.

(2 points)

b) Create a class `Manager` that extends `Employee`. The `Manager` class should have an additional field:

- `int teamSize`

Implement getter and setter methods for this field. Override the `calculateBonus()` method to return a bonus equal to 10% of the salary, plus an additional 5% for each employee in the team (use the `teamSize` field).

(1 points)

c) Create a class `Developer` that also extends `Employee`. The `Developer` class should have an additional field:

- `String programmingLanguage`

Implement getter and setter methods for this field. Override the `calculateBonus()` method to return a bonus equal to 15% of the salary for developers who know the language `Java` and 10% for others.

(1 points)

d) In the main program, create instances of both `Manager` and `Developer`. Set the fields for each using the constructor and setter methods. Calculate and print their bonuses.

(2 points)

e) On the next page, you'll find a small C++ test program. Your task is to analyze the code and predict its output without running it. Pay close attention to virtual function calls and polymorphism. (Note: You can ignore any memory leaks in this example.)

```cpp
class X {
    public:
    virtual void g1() {
        g2();
        cout << "X - g1" << endl;
    }
    virtual void g2() {
        cout << "X - g2" << endl;
    }
};

class Y : public X {
    public:
    void g1() override {
        g2();
        cout << "Y - g1" << endl;
    }
    void g2() override {
        cout << "Y - g2" << endl;
    }
    virtual void g3() {
        g1();
        cout << "Y - g3" << endl;
    }
};

class Z : public Y {
    public:
    void g1() override {
        g2();
        cout << "Z - g1" << endl;
    }
    virtual void g4() {
        g1();
        cout << "Z - g4" << endl;
    }
};

class W : public Z {
    public:
    void g1() override {
        g2();
        cout << "W - g1" << endl;
    }
    void g2() override {
        cout << "W - g2" << endl;
    }
};

int main() {
    X* objX = new X();
    Y* objY = new Y();
    X* refY = objY;
    Z* objZ = new Z();
    X* refW = new W();

    objX->g1(); cout << "*********" << endl;
    objY->g1(); cout << "*********" << endl;
    objY->g2(); cout << "*********" << endl;
    refY->g2(); cout << "*********" << endl;
    objY->g3(); cout << "*********" << endl;
    refW->g1();
    objZ->g4();

    return 0;
}
```

(3 points)

Question 4: Operator Overloading **(10 points)**

In this exercise, you will design a class to represent complex numbers, implement fundamental arithmetic operations, and use operator overloading to enhance code readability and usability.

a) Define a class to represent complex numbers and implement member functions that perform addition, subtraction, multiplication, and division. The function prototype for these operations should be:

```
    COMPLEX performOperation(COMPLEX num1, COMPLEX num2);
```

You can use the following example to test your implementation:

$$(2+4i) - \frac{(6+2i)+(1-3i)}{(3-5i)} = (0.8+2.4i)$$

(4 points)

b) A cleaner and more intuitive approach would be to overload the relevant operators in your class to make operations more natural. The expected usage should look like this:

```
COMPLEX a, b;

cout << "Testing Operator Overloading: " << endl;
cout << "----------------------------------------" << endl;
cout << "Addition:        x + y = " << x + y << endl;
cout << "Subtraction:     x - y = " << x - y << endl << endl;
cout << "Multiplication:  x * y = " << x * y << endl;
cout << "Division:        x / y = " << x / y << endl;
cout << "Conjugation:     x = " << x << " and x! = " << !x << endl;
```

Ensure that all four arithmetic operations are implemented via operator overloading. Additionally, implement a unary operator to compute the conjugate of a complex number. To properly display complex numbers in the console, overload the `<<` operator.

(6 points)

Question 5: C++ Templates (4 points)

a) In this exercise, you will implement a generic `Queue` class using C++ templates. The motivation behind this task is to allow the creation of queues for different data types without duplicating code. Instead of writing a separate queue class for integers, doubles, or strings, you will design a single, reusable template-based queue.

Implement a `Queue` class using templates in C++. Your class should provide the following functionalities:

- Enqueue elements to the back of the queue.
- Dequeue elements from the front of the queue.
- Check if the queue is empty.
- Get the front element without removing it.
- Print the queue contents.

Write a main function that demonstrates your `Queue` class by instantiating a queue of integers and a queue of strings.

Optional Task: (no points again though)

In previous exercises, you have worked with stacks and priority queues. Now, extend your template implementation by creating a `Deque` (double-ended queue) class using templates. Your Deque class should:

- Support insertion and removal from both the front and the back.
- Be implemented using a doubly linked list.
- Provide functions for checking if it is empty, retrieving the front and back elements, and printing its contents.

Write a test program to showcase your `Deque` class by using it with both integers and floating-point numbers.

(4 points)

Question 6: Tic-Tac-Toe (Python) (10 points)

In this task you have to implement the game Tic-Tac-Toe [1] on the command line written **entirely** in the Python programming language.

Write a Python program that allows you to play the game on the command line.

Your program should have the following features:

- Starting player is chosen at random.
- If the board is filled completely without a winner, the game starts anew with the opposite player starting.
- Player can choose whether to play with another human player or against a computer AI.
- How you implement the AI is up to you, at its simplest you can make the computer play at random.
- Game conditions like restart, quit game, change mode (AI or 2nd player) should be implemented.
- Inputting the field of choice can be done via the numpad, e.g. for the lower left field you can press `1+ENTER` and for the middle field you press `5+ENTER` etc (sum of row and column number).

Game session start :

```
Turn of Player X
    1   2   3
  -------------
6 |   |   |   |
  ----+---+----
3 |   |   |   |
  ----+---+----
0 |   |   |   |
  -------------
```

Within a game session:

```
Turn of Player O
    1   2   3
  -------------
6 |   | X | O |
  ----+---+----
3 |   |   | X |
  ----+---+----
0 |   |   |   |
  -------------
```

Question 7: The Ultimate Game (Optional) (0 points)

In addition to the grand sum of 0 points, you will also earn the Tutor's *mad respect* for completing this optional exercise.

Extend your previous game implementation with your own GUI or add colors to the command line to make it look prettier. For the GUI you can use Tkinter, the go-to GUI framework for python.
`https://en.wikipedia.org/wiki/Tkinter`
`https://tkdocs.com/tutorial/index.html`

[1] **https://en.wikipedia.org/wiki/Tic-tac-toe**