UNIVERSITÄT BASEL

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

UNI BASEL

Programming Paradigms – C++ FS 2022

Exercise 2 Due: 02.05.2022 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**. 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 (9 points)

Implement an ordered set of data-elements, each with a unique integer identifier, 2 float values and a string name. To this avail, sort the elements by the Euclidean norm of the two floats ($\sqrt{x^2 + y^2}$) in natural ascending order (e.g. $1.2 < 2.4 < 3.2$). Also, check the uniqueness of each element on insertion or resolve the uniqueness another way. The sorting function must be part of the `Set`-class.

You are free to chose an underlying data structure as the basis of your implementation. We suggest using a simple array-structure or double-linked list. If you don't mind the effort, you can consider alternatives. For instance, you might want to use a *binary search tree* (BST) as the underlying data structure to get a better performing implementation. The latter also allows to implement a dynamically growing set.

The class should contain the following fuction signatures:

```
/*
 * An insert function. The function returns 1 if the
 * integer was added, and 0 if it was already in the set.
 * If the default value "-1" is used, i.e. no ID is provided
 */
int add( double x , double y , string name , int id = -1 );

/*
 * A removal function. The function returns 1 if the
 * integer was removed and 0 if it was not in the set.
 */
int remove( int id );
/*
 * Bonus : Add a function allowing to remove elements with other
 *         criteria like order-value, value of floats or name.
 */

/*
 * A test function. The function returns 1 if the set contains
 * the integer and 0 otherwise.
 */
int contains( int id );
/*
 * Bonus : Add a function allowing to check for elements with other
 *         criteria like order-value, value of floats or name.
 */

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

/*
 * A sort function. You may freely chose a comparative
 * sort algorithm such as Merge Sort for your implementation.
 * @var reorganise : defines whether the data structure is changed
 *                   or a fasts access sort-list is used.
 */
void sort( int sort_type = 0 , bool reorganise = false );
/*
 * Bonus : Allow external sort calls for other parameters, i.e.
 *         sort by name.
 *         Use enums for the sorting type.
 */


/*
```

```
48  * A function that outputs all integers in the set on the
49  * console, delimited by commas.
50  */
51 void print();
52
53 /*
54  * A function that releases allocated memory. Make sure,
55  * that after this function call, allocated memory was
56  * correctly deleted. The difference to the destructor is
57  * that the object still exists but reset to initialisation.
58  */
59 void destroy();
```

Write a program to test each function of your implementation.

Question 2: C++ Classes I : Constructor, Destructor (12 points)

In this task, we will focus on C++ classes and their construction, i.e. instantiation, and their destruction, i.e. unassignment of memory space. As non-primitive non-simplistic objects, these actions have to be handled manually and explicitly (compared to simplistic non-primitive objects like arrays, data-fields and simple `structs`). To test this handling, you will implement a dynamic array with a manually implemented write buffer.

A dynamic array is basically a data-field of uniform elements, whose size can be dynamically changed during its life-time. Hence, it possesses a pointer linked to a data-field with excess memory, reserved for not yet inserted elements. This additionally available memory is kept in a suitable size. In other words, it allows common additions without the necessity of changing memory reservation all the time. Yet, it does not block unnecessary large memory segments, i.e. gets suitably reduced.
As a managed dynamic array, this is executed by the array-object itself without concerning the library user.
Another performance boost technique are write buffers. For large data sets or sets with large elements, consecutive operations are buffered to not access slow memory each time. The buffer is a priority list that stores changes, whose executions are paused until the next suitable timing. Typically, such timing is a change of available memory, reaching a buffer limit (i.e. maximal number of buffered operations) or when the function trim() is manually called.

Please note, that such optimisation like excess memory and buffering, also add an overhead (time loss) and are commonly only used if required, i.e. advantageous. For the purpose of this exercise, we use them anyway with simple integer values.

Also, do not use pre-implemented libraries like the `vector` C++ class. On a side node, the latter is such a managed dynamic array several optimisations.

General Hint : To discuss different C++ language mechanics, some structure repetition is expected. To reduce coding time, we suggest reusing, i.e. copy-pasting, suitable parts between questions.

a) Basic Object

Declare a class `DynamicArray` in a header file with

- a user-defined constructor & destructor
- `get` function (return element)
- `set` function (change element)
- `add` function
- `remove` function
- `trim` function
- a `data` pointer variable
- a `size` int variable (number of elements)
- an `avail` int variable (available storage in elements)
- a `buffer` pointer variable (priority list)
- an `access` pointer field (static array of tuples of elements with changes buffered and a direct link to the its latest value or relevant sequence of changes in the buffer)

Declare a class `BufferedChange` in the same header file as before, containing:

- an `op` integer variable defining the type of allowed changes (add, subtract, overwrite)
- a pointer variable `task` that points to an element.
- a `value` integer value of the connected change (e.g. subtract 3 renders `value`=3 and `op`=subtract)
- a function for every allowed operator
- an `execute` function to apply changes
- a `getValueOf` function to get the most recent value of a given element (base value given by pointer)
- a user-defined constructor and destructor

(1 points)

b) Implement a Managed Dynamic Array

Implement `DynamicArray`, containing the following methods and variables:

- Allocate and initialise basic variables in the constructor.
- When adding or removing elements, check whether the available memory is suitable. Allocate more memory, once `avail` $= 5/4 \cdot$ `size` and free memory, once `avail` $> 2 \cdot$ `size`.
- When changing the available memory, a new data-field is allocated, changes are executed, data is moved and the old memory is freed.
- `trim` executes the buffered changes at sets the available memory to $3/2$
- `get` is the only access to stored elements
- `get` returns the most recent value that includes all buffered changes
- `set` supports only the allowed operations (same as defined in `ChangeBuffer`)
- You can use an enumerator to define set operations.
- Only changes of values are buffered, not the insertion/removal of elements.
- The destructor must clean-up **all** data linked to this instance of the object.
- The `ChangeBuffer` priority list is defined as a linked list. Any addition is appended using the first element (initialised 0-operation).
- For faster access, any changed element with buffered changes gets an entry is the `access` list.
- The `access` list is a list of tuples, an element identifier (pointer or number) and a pointer to the most recent value of/sequence of changes to that element.
- The `access` list has 10 entries, which is the maximum allowed number of buffered changes.

(4 points)

c) Implement a Write Buffer

Implement the `BufferedChange`. For simplicity, we suggest using a single-linked list, containing the following methods and variables:

- An `append` function that adds a new task to the end of the linked list.
- The method `append` returns a pointer to the beginning of latest relevant chain of operation for an element. Note that an `overwrite` operation removes dependencies to previous changes.
- An `execute` function that performs all buffered changes in sequence (removing them from the list).
- The `execute` function does not delete the 0-operation element that exists only at the root of the linked list.
- A `getValueOf` function that follows the linked list, updating a copy of the stored element value given by the pointer to the store element, to its most recent value.
- Execution functions for all allowed operations, receiving a pointer to the element to be changed and a connected value variable.
- A Constructor and destructor (With **clean** allocation/deallocation)

Bonus Task (0 Points) : Instead of an explanation do the implementation.

(4 points)

d) Test Functions

Write an executable that instanciates the implemented managed dynamic list with write-buffer and tests it.

- The test should execute all available operations multiple times in different order and combination.
- Every function has to be called at least once.
- Use 40 entries at the beginning and add/remove 5 and 50 to test all cases.

(3 points)

Question 3: C++ Classes II : Inheritance (13 points)

With classes and inheritance, the C++ language allows programmers to remove code duplication, implement simple conversions, or easy modularisation. A typical example for this are data classes, i.e. data containers with handlers, that are moved between parts and modules of a program. In the following, you will implement this concept application.

In the following, you expand the functionality step-wise with child-classes. Although you can use the given hierarchy, it is up to you to choose whether you want to use a different one and move functionality between the levels of inheritance. In any case, argue your decision.

a) The Base Class

Create a header file `data.h` declaring a class `Data`. This class shall contain :

- basic data holders, i.e. list or array, of `ElementType`
- other elementary variables necessary for general data description
- Basic functions that perform general actions on, from and to data (at least `get`, `delete` , `add` and `empty`).
- Functions depending on functionality given in child classes (extensions), can be defined virtually or even purly abstract to be referred to early on. This concept is equivalent with interfaces and abstractions with Java classes.

A second class called `ElementType` that contains :

- a `string` variable as descriptor
- an unique identifier of choosable type
- a list/array of 10 `double` values, defining properties
- a list of `string`s that are used to categories the element or describe its composition (number of elements can vary)
- some explicit inline operations can be defined here too (please argue why)

The actual choice of optimal declarations are left to you yourself, and shall be based upon or extended on the following sub-tasks. For each choice add a short reasoning, why it is declared in this base class.

(2 points)

b) Sort and Search

Expand your Class(es).

Create a sort function that lets you sort the data entries in ascending and descending order by a property value or the sum over all property values (as 11th property).

Implement two sorting modes, one optimised for search and another optimised for fast access. With some thought and not especially aiming for performance, you can use simplistic and compact implementations. If you don't mind the effort, you can also consider more elaborate versions. For search, this means Tree-Structure or Heap-Structure for instance. For access, this means block-structures with overlying heap-structure for further optimisation.

```
1 void sort_assending_search(int parameter_no);
2 void sort_assending_access(int parameter_no);
3 void sort_descending_search(int parameter_no);
4 void sort_descending_access(int parameter_no);
```

Hint : Additional or replacing data containers can be used with linked or duplicated (partial) data. Ensure by overloading access functions. Hint : You can use enums with a single `sort(...)` call instead, if you like.

(4 points)

c) Bonus Task : Sort and Search 2

Add additional sorting functionality for the other element parameters, each with a fast search and fast access variant.

Use a B-tree or another suitable data structure for the more complex data like the component strings.

Add additional functionality like sorting by more than one parameter hierarchically or independently (multiple search lists).

(0 points)

d) Operator Overload

Expand your Class(es).

Overwrite the get element []-operator to fast access the property values of entries to be used in matrix calculations.

Overwrite the (−)-operation between two data elements, where a new unique identifier is created, properties are handled numerically and categories/compositions show the differences (remove intersection).

Add a `mean_diff` function that reduces all elements to be expressed by their difference to a mean element and write a getter for that element.

```
[elementType] [elementType]::operator[](const int &no)
[elementType] [elementType]::operator-(
                                const [elementType] &rightside);
void [dataSort]::mean_diff();
```

You can name `[dataSort]` and `[elementType]` how you like.

(3 points)

e) Functors

Write a mathematical **base change** for the property values as a functor that can be given to an external call (like for processing data by a GPU or another accelerator). Write functions for both directions.

Use conversion "[element] conversion" : [0] erg→Joul, [1-3] feet→meter, [4] °F→°C, [5-6] PS→kW, [7-9] inc (Euklidean)→m (Polar)

```
elementType convertA2B(const elementType &in);
elementType convertB2A(const elementType &in);
```

Hint: Keep accessibility of fields in mind. Friend functions or public declarations could help.

(1 points)

f) Bonus Task : copy VS asign

Implement and explain the difference in memory handling between the the previous implementation and the following variations:

```
void convertA2B(elementType &in);
void convertB2A(elementType &in);
void convertA2B_bad(elementType in);
void convertB2A_bad(elementType in);
```

(0 points)

g) IOData 1

Expand your Class(es).

Writing to the file in C++ can be done via streams. Creating an output stream, resp. an input stream, allows you to write to a file in a identically manner like writing to the terminal with "`cout`", resp. reading from the terminal with "`cin`". The differences are that you use file-streams instead of basic IO-streams, and you have to open and close a file. [1]

Overwrite the output stream Operator "<<"/">>" to print the data to terminal using "cout" or to a file. (Attention on stream direction!)

Write a "print()" function that writes the output-stream to a file.

Choose a suitable format (e.g. with or without unit names with values , (sub-)grouping or not , title line or not , ...). Keep readability or post-processing in mind. Briefly explain your choices.

(1 points)

h) Bonus Task : IOData 2

Add a read in functionality to your class.

Overwrite the input stream Operator ">>"/"<<" to use "cin" or an input file.

(0 points)

i) Bonus Task : Field-Inheritance

Expand the `ElementType` class with a new child and use it with the `Data` class. Implement and explain the necessary changes, or explain why none are needed.

Hint : Depending on your implementation of a previous subtask, you already implemented a field-class child extension.

(0 points)

j) Bonus Task : Diamond Problem

Up to now, we considered inheritance as a singular chain, each element extending the functionality of the prior class. Instead, consider each extension separately used independently when necessary. Hence, for instantiation we have any combinations of extensions available. Some of these combinations share some base classes. Using two or more of them as extension is referred to as "diamond problem".

Explain the relationship and casting properties between these combinations and extension combining all of them, i.e. the last child of the original single chain inheritance. Modify you current implementation to allow the instantiation of these combinations and test it out yourself.

E.g. : ((Data → sort) + (Data → operable)) → oparable sorted Data

(0 points)

[1] See **https://www.cplusplus.com/doc/tutorial/files/**

k) Testing

Write a test function with a sub-routine for each task taking the most general level of the data-object (highest parent) as parameter.

- Inputs 10 elements to your data-object using the provided list (see data.txt) .
- Sort values by an entry of your choice in acceding order for fast access and print out the list. The generality level of the data-object should be the maximal supporting one.
- Sort values by an entry of your choice in descending order for fast search and print out the 3rd smallest value.
- Apply a 10x10 matrix multiplication to all property data of all elements. Choose the following matrix for simple verification.

$$\begin{pmatrix} 1 & 0.5 & 0.5 & 0.5 & 3 & 4 & 4 & 0.3 & 0.2 & 0.3 \\ 0 & \cos\left(\frac{\pi}{4}\right) & -\sin\left(\frac{\pi}{4}\right) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sin\left(\frac{\pi}{4}\right) & \cos\left(\frac{\pi}{4}\right) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 12.45 & 12.45 & 12.45 & 100 & 0 & 0 & -11.1 & -11.1 & -11.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}^T$$

  Note : Do not assume a sparse matrix, this is just used for easier validation.
- Print out the data-set three times, one before, one after the conversion and one with values relative to mean (mean_diff).
- Design additional tests for executed bonus tasks.

Write a main function that launches all tests in sequence

(2 points)

Question 4: C++ Templates (4 points)

Templates are an other generalisation technique in C++ other than base classes with inheritance. In difference to class generalisation that varies mostly functional implementation, template vary variable and member types. It does not matter whether these variables are used as arguments, return values, field-types or local variables.

For this, we consider a sorted dynamic array (you can choose yourself if with or without write buffer) :

- Choose freely your sorting method of implementation (part of the class).
- implemented change operations are addition, subtraction, multiplication, division
- the sorting is updated after an element got changed
- plan for element types to be short int, int, long int, long long int, short float, float, double, long double, ...
  Hence, you can assume existing "¡" , "=" , "+" , "-" , "*" , "/" operations on elements.
- Hint : You can use one of the previous task implementations as foundation.

a) Implementation

Implement the `BSClass` as template of the underlying element type.

(3 points)

b) Bonus Task : Implementation II

Change your implementation to also allow other types like string. With operation undefined for types like this, change operation methods to be virtual or even abstract (pure virtual).

Hint : For templates with virtual functions in C++ check out "Policy Based Design" and "Implementer" type function-pointer.

(0 points)

c) Test

Test your implementation implementing a main function instantiating as byte (8-bit int) , long long integer (64-bit int) , float (32-bit floating-point) and long double (80-128-bit[2] floating-point). Fill with 100 random values and also test overflow with smaller types.

With implemented Bonus Task, also test with this instantiation.

(1 points)

[2]mostly CPU, but some times also compiler and OS specific

Question 5: 4-in-a-Row (12 points)

In this task you have to implement "four in a row" [3] 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:

- The entire board is 9 times 9 elements.
- Each player can in alternating turns choose a column and add dot of is "color" to its lowest empty element.
- The player adding the last dot to the field to create 4 dots of equal "color" in a line wins.
- Allowed conditions for a line are horizontal, vertical and diagonal.
- Starting player is chosen at random.
- if the board is filled completely without a winner, the game starts anew with the opposite player starting.

Game session start :

```
Turn of Player o
   A B C D E F G H I
8  . . . . . . . . .
7  . . . . . . . . .
6  . . . . . . . . .
5  . . . . . . . . .
4  . . . . . . . . .
3  . . . . . . . . .
2  . . . . . . . . .
1  . . . . . . . . .
0  . . . . . . . . .
```

Within a game session:

```
Turn of Player X
   A B C D E F G H I
8  . . . . . . . . .
7  . . . . . . . . .
6  . . . . . . . . .
5  . . . . . . . . .
4  . . . . . . . . .
3  . . . . X o . . .
2  . . . o X X o . .
1  . . o X o o X . .
0  . . X X o o X o .
```

If you need help to get started with Python, take a look at the Python tutorial:
`https://docs.python.org/3/tutorial/index.html`

[3] **https://en.wikipedia.org/wiki/Connect_Four**

Question 6: The Ultimate Game (Optional) (0 points)

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

Extend your previous game implementation such that you introduce a single player mode with a AI based on randomness or surprise us.

Also, use colors on the command line instead of the passionless white on black that is standard. Or, if you are really ambitious, you may even code a GUI for the game. Be creative!