Chair for System Simulation

Winter Term 2023/2024

Prof. Dr. Harald Köstler, Richard Angersbach

Advanced Programming Techniques
Sheet 1 — First Steps

This sheet is not mandatory. It is only a preparation for the programming project later in the semester. However, we strongly recommend you work on all these exercises.

Part 1 — Environment and Warm-Up

Please be aware that during the exercises of AdvPT, we will only provide support for the Linux machines in the CIP-Pool. You can use them remotely with ssh. If you have never worked with the Linux terminal, we recommend going through some tutorials first.

Furthermore, we expect you to know essentials like the difference between stack and heap or the use of pointers. We also recommend you to learn about Git, Make, and CMake. These tools are invaluable for C++ development. If you never worked with C/C++ we recommend watching The Structure of a Program and Pointers and Memory.

Here are some links regarding the mentioned topics. Feel free to explore these subjects on your own with your favorite search engine.

- Linux Tutorial
- Linux Terminal
- Stack vs Heap
- CMake
- Make Tutorial
- Valgrind
- gdb Tutorial

a) Range Sum Write a program that queries the user for two numbers and sums the numbers in that range (including the first number, excluding the last number).

b) Factorial Write a program that prompts the user to enter a number and then calculates the approximate factorial of the given number as a double precision floating point number and writes it to the standard output. Verify your program at least against the following *test cases*:

$$0! = 1.0 \quad ; \quad 1! = 1.0 \quad ; \quad 6! = 720.0 \quad ; \quad 12! = 479001600.0$$
$$13! = 6227020800.0 \quad ; \quad 21! \approx 5.1091e19$$
$$35! \approx 1.0333e40 \quad ; \quad -1! = ?$$

c) Punctuation Write a program that reads a line from standard input and prints the line to standard output but with all punctuation removed. The resulting program should be usable as a filter like this:

```
./punctuation < with_punct.txt > no_punct.txt
```

Hint: Have a look at the following STL Header: `<cctype>`

Part 2 — C++ Variables and Basic Types

Complete the Studon Test “C++ Variables and Basic Types”.

Part 3 — Function Resolution

Complete the following Studon tests:

a) Function overloading Demonstrate your basic knowledge in the resolution of functions by taking a quick StudOn Quiz (“Function resolution - basics”).

b) Inherited functions Demonstrate your advanced knowledge in the resolution of functions by taking a quick StudOn Quiz (“Function resolution - classes and structs”).

c) Template function specialization Demonstrate your even more advanced knowledge in the resolution of functions by taking a quick StudOn Quiz (“Function resolution - templates”).

d) Mixed concepts Show off your extreme knowledge in the resolution of functions by taking a quick StudOn Quiz (“Function resolution - challenge”).

Part 4 — Multidimensional arrays / Tensors

In this exercise, you will write a simple multidimensional array data structure. Its *shape* is a tuple of positive integers that defines how many elements are stored in each direction. The resulting implementation shall resemble a simplified version of numpy’s ndarray.

Multidimensional arrays are crucial for computations in numerical linear algebra. In particular they are the workhorse data structure in software that is used to approximate partial differential equations and their solutions as well as in machine learning applications. The latter usually uses the term *tensor* to refer to the same data structure[1]. Clearly, every engineer or researcher should be able to write their tools from scratch before using them in production. So we will do that now.

Before working on this exercise, you should download and extract the corresponding files from StudOn.

Compile the exercise with CMake and Make. We ship the exercises with a main function and tests. Eventually you can run the tests to verify your implementation. We are using `Wall`, `Wextra`, `Werror` and `pedantic` flags. These flags should enforce you to write clean code.

Thorough out the exercise, there will be links to other resources and to the C++ Core Guidelines. Please follow the guidelines, as they immensely help with writing solid code.

a) Tensor class-template Implement your very own Tensor class-template, including the following features:

- arbitrary shape (with an arbitrary number of dimensions),
- arbitrary component type (to instantiate integer tensors, floating-point tensors, etc.),
- copy- and move-constructors and -assignment operators,
- element accessors,
- (de-)serialization (reading and writing tensors to files).

A skeleton is given in `tensor.hpp`. Regrettably, the exercise might be ahead of the lecture on some topics. However, we think that teaching you modern C++ with the correct habits is more important than convenience. (We ignore guideline C.100 for this educational task.)

[1] Have a look at the tensor classes in TensorFlow and PyTorch.

Subtasks

- *Structure.* Think about how you want to implement the `Tensor` class-template, especially what member variables your class should have. Add these member variables in `tensor.hpp`. You can add (private) helper methods if you want to. You are only allowed to change `tensor.hpp`. The changes must work with the other provided files.
  - C.9
  - C.12
- *Concepts.* The class template is associated with a constraint using the concepts feature introduced in C++20 (see cppreference). Tensors shall only be allowed to hold arithmetic types (e.g. `int`, `float`).
  - T.concepts
- *Constructors.* Implement a primary constructor that receives the shape of the tensor. The shape defines the number of elements in each dimension. An empty shape vector results in a scalar, a shape of size 1 is equivalent to a vector with `shape[0]` elements. A shape of size 2 results in a tensor that resembles a matrix. The number of dimensions (or size of the shape-vector) is also referred to as *rank*. The default constructor shall create a tensor with rank 0, containing a single, zero-initialized entry. Use `std::fill` to fill up the memory with the `fillValue` if specified.
  - C.40
  - C.41
- *Copy-constructor.* Implement the copy-constructor, that takes a reference to an existing `Tensor`, and returns a `Tensor` with the same shape and contents. Use `std::copy` for copying the contents.
  - C.21
  - C.61
  - C.101
- *Move-constructor.* Implement the move-constructor, too. The move-constructor behaves similarly to the copy-constructor, except that it may reuse the storage of the `Tensor` whose rvalue reference we receive.

  std::move is C++'s answer on having very stable resource management while retaining high performance. It is crucial that you understand the difference between `lvalues` and `rvalues`. Move Semantics is a tremendous introduction to this topic.
  - C.64
  - C.66
  - C.102

  `std::exchange` makes a highly readable code.

  After moving, the moved rvalue shall be equal to a default-constructed tensor.
- *Destructor.* Implement the destructor. Make sure to free any resources that you allocated in the constructor.
  - C.30
  - C.31
  - C.32
  - C.33

  - C.36

- *Assignment.* Implement the assignment operator and a move assignment operator.
  - C.62
  - C.65

- *Accessors.* Implement `operator()` for read and write access to the tensor elements. The element index is passed as a vector of size `rank`. Indexing starts at 0.

- *Member functions.* Implement the `rank`, `shape`, and `numElements` member functions.
  - C.4

- *File IO.* Implement the functions `writeTensorToFile` and `readTensorFromFile`. The file format is defined as follows:

```
<rank>
<shape[0]>
...
<shape[rank-1]>
<t(0, 0, ..., 0, 0)>
<t(0, 0, ..., 0, 1)>
...
<t(0, 0, ..., 0, shape[rank-1] - 1)>
<t(0, 0, ..., 1, 0)>
<t(0, 0, ..., 1, 1)>
...
<t(shape[0] - 1, shape[1] - 1, ..., shape[rank-1] - 1)>
```

  First the `rank` (i.e. size of the shape array) is specified. Next all `rank-1` entries of the shape array. Eventually all elements of the tensor follow. All entries are separated by spaces.

- *Testing.* Make sure that all the tests in `test_tensor.cpp` work with your implementation.

- *Memory leaks.* Use valgrind[2] to check for memory leaks.

- *Readability.* Check your code. Can you enhance the readability?

b) Matrix-vector multiplication Many applications frequently use matrices and vectors. So it would be convenient to specialize our tensor class-template for that purpose. Use the tensor class-template to write your own matrix and vector class-templates. The skeletons are supplied in `matvec.hpp`. This should be quite straightforward as many features can be reused from the tensor implementation. Then implement matrix-vector multiplication and make sure that all tests in `test_matvec.cpp` pass.

[2] `valgrind ./test_tensor -leak-check=full`