Chair for System Simulation

Winter Term 2021/2022

Prof. Dr. Harald Köstler, Jan Hönig, Marco Heisig

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.

Even though this sheet is not mandatory, we will grade your submissions. The purpose of this is to give you feedback and help you improve your skills.

Exercise 1 — Warm-Up

Please be aware that during the exercises of AdvPT, we will give only 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 by Example
- 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>`

Exercise 2 — Escaping the Grid

In this exercise, you will write a program that finds paths through a simple, two-dimensional grid. 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. 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 C++ Core Guidelines. Please follow the guidelines, as they immensely help with writing solid code.

a) My Very Own Grid Implement your very own two-dimensional grid called `MyGrid`.

Motivation

> C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off – Bjarne Stroustrup

Why on earth do we want you to implement a simple 2D-Array Class when the standard library offers multiple options like `vector` and `array`? C++ is well known to be a fast language, but also a dangerous one. Unlike other languages, in C++ you need to handle resources yourself, including memory. In this exercise, you will learn how to treat resources safely. On top of them, you will use basic inheritance and operator overloading. 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 the C.100 for this educational task.)

Task

- Think about how you want to implement the `MyGrid`, especially what member variables your class should have. Add these member variables in `MyGrid.hpp`. You can also add (private) helper functions if you desire to. You are only allowed to change `MyGrid.hpp` and `MyGrid.cpp`. The changes must work with the other provided files.
  - C.9
  - C.12
- Implement a primary constructor that receives three arguments — the number of rows, the number of columns, and a default tile — and that returns a new `MyGrid` instance. Use `std::fill` to fill up the memory with the `initalTile`.
  - C.40
  - C.41
- Implement the copy constructor that takes a reference to an existing `MyGrid`, and that returns a `MyGrid` with the same size and contents. Use `std::copy` for copying the contents.
  - C.21
  - C.61
  - C.101
- Implement the move constructor, too. The move constructor behaves similarly to the copy constructor, except that it may reuse the storage of the `Grid` 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 readible code.

- 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

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

- Implement the `rows` and `cols` member functions.
  - C.4

- Implement the `validPosition` member function, and then use it to implement the two `()`-operators. Make sure that your `()`-operators throw an exception if the access is outside of the valid range of rows and columns.
  - E.2
  - E.3

- Implement the `operator«` function. It should write the number of rows in the first line, the number of columns in the second line, and then each row of the grid in a separate line with one character per Tile. Use the auxiliary functions `tile_from_char` and `char_from_tile` to convert between tiles and characters.

- Implement the `read` static member function. It should be able to turn a text representation like the one produced by `print` back into a `MyGrid` instance.

- Make sure that all the tests in `gridtest.cpp` work with your implementation. Use valgrind[1] to check for memory leaks.

- Instead of `new[]` and `delete[]` make use of a `std::unique_ptr<Tile[]>` to ensure ownership of the allocated memory. Smart Pointers gives you a great introduction, what smart pointers such as `std::unique_ptr` are. This shoul enable you to use the default destructor. What about the move constructor and move assginments, can we use the `default` there? What about the copy constructor and copy assignment, do we have to change them? (Exceptions gives you a nice introduction into Exceptions, which is currently a little off-topic, however Klaus Iglberger has a great example of Copy/Move constructor/assignment relation from 26 to 49 minutes of the video)
  - R.3

- Check your code. Can you enhance the readability?

If you want us to grade this exercise, you need to upload your `MyGrid.hpp` and `MyGrid.cpp` files (individually or as a zip file) on StudOn.

[1] `valgrind ./gridtest -leak-check=full`

b) Path Finding In this second task, you have to write a program that can "escape" the grid, by finding a path of floor tiles from the tile in the upper left corner (the tile whose row and column is one) to a boundary tile that is not a wall tile. No diagonal moves are permitted. Once you have found such a valid path, draw it to the grid by changing all tiles on the path to `Path`. To do so, implement the `escape` member function of the `Grid` class in the file `escape.cpp`. Don't forget that you can use the entire C++ standard library here.

For the test grid `C` that was included in the project skeleton you have received via Studon, your solution should look like this:

```
8
8
########
#******.#
#.###*##
#.#.#*##
#.#..**#
#.####*#
#....#*#
######*#
```

If you want us to grade this exercise, you need to upload your `escape.cpp` file (individually or as a zip file) on Studon.
Extra challenge: Can you write your program so that it always finds one of the shortest paths out of the room?

Exercise 3 — C++ Variables and Basic Types

Complete the Studon Test "C++ Variables and Basic Types".

Exercise 4 — 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").