Systems Engineering and Data Processing in C++
Summer Term 2021

Hasso Plattner Institute
Computer Graphics Systems Group

Prof. Dr. Jürgen Döllner
and Willy Scheibel

Assignments #3: Custom Types and STL Compatibility

Objectives

In these assignments you will learn how to:

- implement custom encoding for I/O streams;
- define types compatible with stdlib types;
- implement custom iterators; and
- design and implement custom types.

Rewards

For each successfully solved task you and your partner (optional) gain one bonus exam point. A serious shortcoming in fulfilling the requirements of a task will result in no bonus point regarding this task. You can gain at most 4 bonus points through exercise sheet.

Submission

All described artifacts has to be submitted to the course moodle system[1] as a zipped archive. The naming convention includes the assignment number and your *personal assessment numbers* with following naming convention: `assignment_3_paNr1.zip` or `assignment_3_paNr1_paNr2.zip` whether or not pair programming was applied. Compiled, intermediate, or temporary files should not be included. If pair programming is used, the results are turned in only once. The assignments #3 are due to the next tutorial on June 3rd, 9:15 a.m. GMT+2.

Presentation

Feel free to present any of your solutions during the Zoom meeting in the upcoming tutorial. If you want to present your solution of a task, please send the source code beforehand via e-mail to willy.scheibel@hpi.de.

General Instructions

Pair Programming On these assignments, you are encouraged (not required) to work with a partner, provided you practice pair programming. Pair programming "is a practice in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, or test." One partner is driving (designing and typing the code) while the other is navigating (reviewing the work, identifying bugs, and asking questions). The two partners switch roles every 30-40 minutes, and on demand, brainstorm.

Cross-Platform The assignments can be solved on all major platforms (i.e., Windows, Linux, macOS). The evaluation of submitted assignments can be carried out on any of these platforms. All results should not use platform-specific dependencies. Platform-specific build and macro code is permitted.

Violation of Rules a violation of rules results in grading the affected assignments with 0 points.

- Writing code with a partner without following the pair programming instructions listed above represents a serious violation of the course collaboration policy.
- Plagiarism represents a serious violation of the course policy.

[1]https://moodle.hpi.de/course/view.php?id=174

Task 3.1: Tiny Encryption for Streams

File: `tea/tea_filebuf.h`, `tea/tea_filebuf.cpp`, `tea/tea_filebuf-test.cpp`, `tea/tea_test1_plain.dat`, `tea/tea_test2_enc.dat`.

Description: The *Tiny Encryption Algorithm* (TEA) is a simple to implement data encryption algorithm; it encodes or decodes 64 bit data using a 128 bit key. Your task is to implement this algorithm for file output streams (encoding) and file input streams (decoding). Use a stream buffer as mechanism to infiltrate TEA-based encoding and decoding.

- Use the TEA reference implementation found on Wikipedia: `https://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm`
- Use the hard-coded TEA key in the framework.
- Adapt the given test cases to use the TEA stream buffer.

Artifacts:

a) `tea_filebuf.h`, `tea_filebuf.cpp`, `tea_filebuf-test.cpp` : Source code of the solution.

Objectives: Customization of I/O streams, stream buffers.

Task 3.2: Custom Type Integration

File: `integration/integration.cpp`.

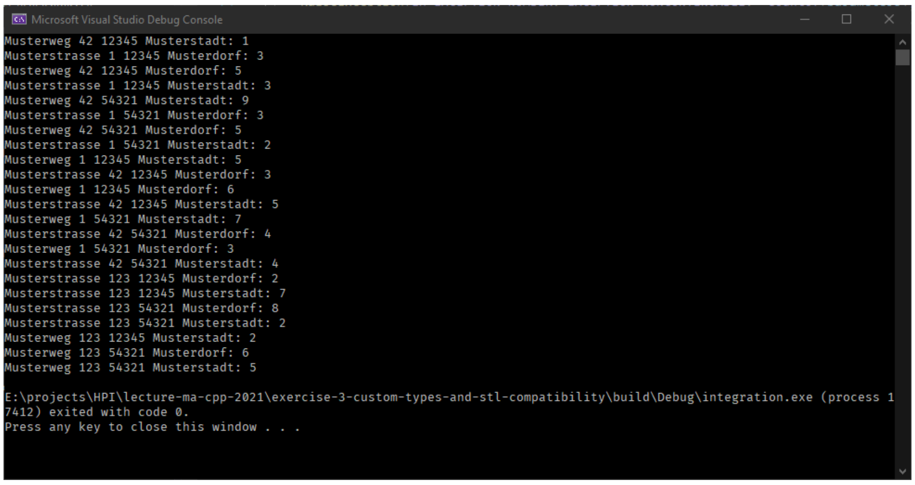
Description: The framework defines the interface for a simple custom type `Address` that is to be used in a variety of language and stdlib constructs; however, the program does not compile because the type does not meet the requirements of these. For example, `std::map` requires that it's key type is comparable and has a total ordering.

Complete the type definition and add all requirements (members and/or external) to make the code work.

Competition: This is a competitive task with respect to memory consumption. If you want to compete in the open competition submit your isolated source code of this task via e-mail to willy.scheibel@hpi.de until June 1st. Your source code is checked for conformity with respect to the core task. The memory consumption of your solution will be measured, not theoretically determined.

You are allowed to define the type's fields using any means available in C++ as long as the four address components are made available via their respective accessor functions. The code of the `main` function must not be changed. You may assume that values do not exceed the ranges present in the data file, but we may use a larger file for measurement.

Example:



```
Musterweg 42 12345 Musterstadt: 1
Musterstrasse 1 12345 Musterdorf: 3
Musterweg 42 12345 Musterdorf: 5
Musterstrasse 1 12345 Musterstadt: 3
Musterweg 42 54321 Musterstadt: 9
Musterstrasse 1 54321 Musterdorf: 3
Musterweg 42 54321 Musterdorf: 5
Musterstrasse 1 54321 Musterstadt: 2
Musterweg 1 12345 Musterstadt: 5
Musterstrasse 42 12345 Musterdorf: 3
Musterweg 1 12345 Musterdorf: 6
Musterstrasse 42 12345 Musterstadt: 5
Musterweg 1 54321 Musterstadt: 7
Musterstrasse 42 54321 Musterdorf: 4
Musterweg 1 54321 Musterdorf: 3
Musterstrasse 42 54321 Musterstadt: 4
Musterstrasse 123 12345 Musterdorf: 2
Musterstrasse 123 12345 Musterstadt: 7
Musterstrasse 123 54321 Musterdorf: 8
Musterstrasse 123 54321 Musterstadt: 2
Musterweg 123 12345 Musterstadt: 2
Musterweg 123 54321 Musterdorf: 6
Musterweg 123 54321 Musterstadt: 5

E:\projects\HPI\lecture-ma-cpp-2021\exercise-3-custom-types-and-stl-compatibility\build\Debug\integration.exe (process 1
7412) exited with code 0.
Press any key to close this window . . .
```

Note: the output order may vary depending on your compiler and implementation.

Artifacts:

a) `integration.cpp` : Source code of the solution.

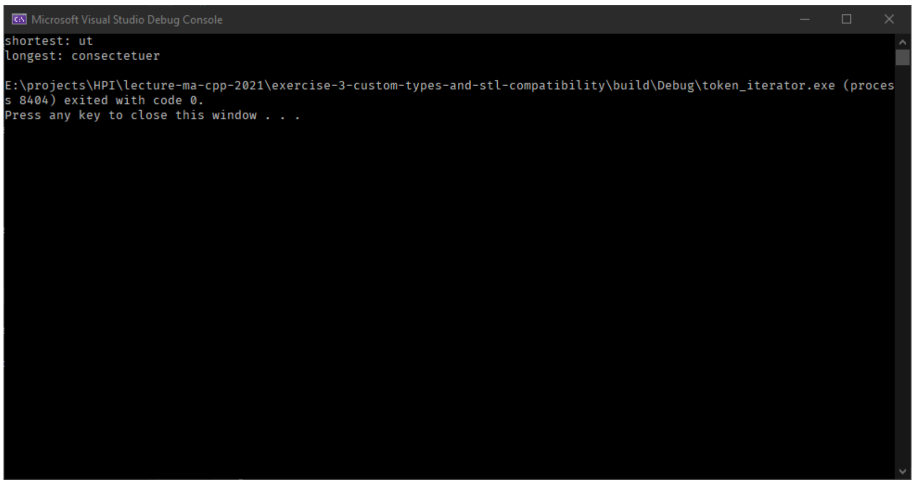
Objectives: Custom types, stdlib integration.

Task 3.3: `std::string_view` Token Iterator

Files:, `token_iterator/token_iterator.cpp`.

Description: Implement tokenization of a large text file with a custom iterator returning `std::string_views`. The iterator type should satisfy the requirements of `std::forward_iterator` (C++20) or *LegacyForwardIterator* (C++17). The returned `string_views` should reference the original memory, such that no copies of the data or parts of it are made.

Example:



```
shortest: ut
longest: consectetuer

E:\projects\HPI\lecture-ma-cpp-2021\exercise-3-custom-types-and-stl-compatibility\build\Debug\token_iterator.exe (proces
s 8404) exited with code 0.
Press any key to close this window . . .
```

Artifacts:

a) `token_iterator.cpp` : Source code of the solution.

Objectives: `std::string_view`, custom iterators.

Task 3.4: Modulo Counters

Files: `modcounter/test.cpp`

Description: Design and implement a concrete type that implements the concept of a modulo counter. A `ModuloCounter` object holds an signed integer value v; the value can incremented and decremented. It has a defined range for its values, i.e., $v \in [min, max)$ (inclusive, exclusive). The value of a module counter is guaranteed to stay in that interval; values are "wrapped" using a modulo operation. The interval is specified as constructor argument; it cannot be modified later. The result of a binary arithmetic operation should have the same type as the first operand and inherit its interval, if applicable.

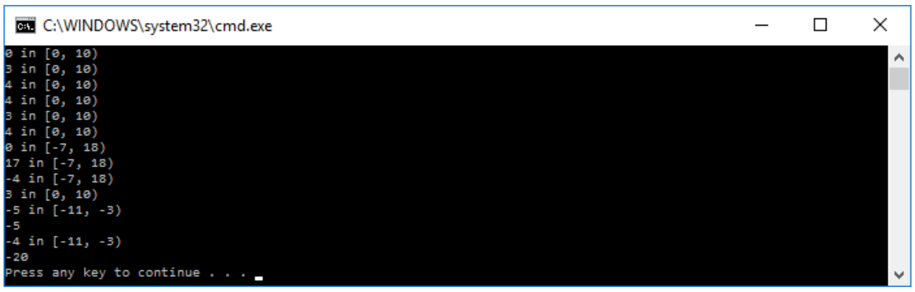
Possible class components include:

- Ordinary constructor
- Copy and move constructors
- Assignment and type conversion operators
- Arithmetic operators
- Get and set operations

You should further extend the class by providing support for streams. Make sure that appropriate assertions are checked. Keep the class as lean as possible but provide "fully featured class" — you can take advantage of the fact that the C++ compiler generates some class elements implicitly. Think about the sematics difference between copy and move constructor for this specific class.

Implement the test case as specified in the comments to produce the output listed below.

Example:



C:\WINDOWS\system32\cmd.exe

```
0 in [0, 10)
3 in [0, 10)
4 in [0, 10)
4 in [0, 10)
3 in [0, 10)
4 in [0, 10)
0 in [-7, 18)
17 in [-7, 18)
-4 in [-7, 18)
3 in [0, 10)
-5 in [-11, -3)
-5
-4 in [-11, -3)
-20
Press any key to continue . . .
```

Artifacts:

a) `modcounter/test.cpp`, `modcounter/...` : Source code of the solution.

Objectives: OOP, concrete classes, interface design, operator overloading