Advanced Programming in C++

Winter Term 2016/17 – Assignments # 4

Hasso Plattner Institute
Computer Graphics Systems Group

Prof. Dr. Jürgen Döllner

Objectives

In the assignments you will learn how to

- implement data structures and functions evaluated at compile time;
- perform inter-process communication and synchronization;
- customize I/O stream behavior by specialized stream buffers.

Task 4.1: Sine Computation using a `constexpr` Function

Files: `sin.cpp`

Description: Trigonometric functions represent one category of costly operations: If used extensively, they can slow down significantly. In addition, the built-in versions computed high-precision results, which is commonly not required by applications.

To provide a fast way to get sine values, implement a lookup table that is constructed at compile-time. To calculate the sine values, define a `constexpr` sine function that calculates a (finite) Taylor series; a parameter can specify the length of the actual series.

- Create and initialize a data structure that represents the lookup table.
- Implement the `fastSin` function to perform the approximate sinus computation based on the lookup table at run-time.
- Single-precision floating point arithmetic is sufficient.

Artifacts:

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

Objectives: Compile-time resolved computations, `constexpr`

Task 4.2: Inter-Process Communication with Memory-Mapped Files

Files: `ipc_viewer.h`, `ipc_viewer.cpp`, `ipc_player.h`, `ipc_player.cpp`, `player.cpp`, `viewer.cpp`.

Description: Implement an inter-process communication (IPC) mechanism for a simple multiplayer game (e.g., dice game). One or more processes represent players that write their scores to a shared file. Simultaneously, a viewer process monitors the contents of the shared file and in this way displays the live score.

Each record in the shared file consists of a unique player ID and the corresponding score (e.g., the sum of all dice rolls for that player).

The scores are stored persistently. The shared file is used each time the game is relaunched.

- Use the `boost::interprocess` library to implement your solution.
- Protect access to the score with appropriate inter-process synchronization primitives from `boost`.
- Implement the three methods defined in the ipc header files as a starting point of your solution. You may add additional files if you decide to split your implementation across multiple header/source files.
- The function specifications included in the header files are part of the assignment description.

The game and its rules are not subject of this assignment as this assignment concentrates on memory-mapped files.

Artifacts:

a) `ipc_viewer.cpp`, `ipc_player.cpp`, `...` : Source code of the solution.

Objectives: Inter-process communication, boost library, concurrency, synchronization

Task 4.3: Stream Redirection using Stream Buffers

Files: `tree_streambuf.h`, `tree_streambuf.cpp`, `tree_streambuf-test.cpp`

Description: Implement a specialized stream buffer `tree_streambuf` that serves as a kind of T-connector for streams. Zero or more instances of `std::streambuf` can be attached to a tree stream buffer, each of which receives the data written to the buffer. Such a buffer can be used, e.g., to mirror console output to a log file.

- Implement a `tree_streambuf` class that inherits from `std::streambuf`.
- Provide a method `void add_sink(std::streambuf *)` to attach stream buffers.
- Adapt the test case to mirror the output written to `std::cout` to the file `cout.log`.
- Ensure that no data can be read from the buffer (write-only).

Artifacts:

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

Objectives: Customization of I/O streams, stream buffers

Task 4.4: Tiny Encryption for Streams

Files: `tea_filebuf.h`, `tea_filebuf.cpp`, `tea_filebuf-test.cpp`, `tea_test1_plain.dat`, `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

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 are required to be cross-platform, that is, they compile successfully and run indifferently with respect to their input and output.

Upload Results All described artifacts are submitted to the course moodle system `https://moodle.hpi3d.de/mod/assignment/view.php?id=2741` as a zipped archive with the following naming convention: `assignment_4_matrikNr1.zip` or `assignment_4_matrikNr1_matrikNr2.zip` whether or not pair programming was applied. Compiled, intermediate, or temporary files should not be included (`*.obj, *.pdb, *.ilk, *.ncb` etc.). If pair programming is used, the results are turned in only once. The assignments #4 are due to the next tutorial on December 14th, 9:15 a.m.

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 (e.g., if one partner does not participate in the process) is a serious violation of the course collaboration policy.
- Plagiarism represents a serious violation of the course policy.