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 #6: Project Setup & Language Integration

Objectives

In these assignments you will learn how to:

- use cmake for advanced project setup,
- use emscripten to cross-compile C++ for Web applications,
- bind scripts into your C++ executables, and
- provide native C++ extensions to script environments.

Rewards

For each successfully solved task you and your partner (optional) gain one bonus exam point. Some tasks allow to choose from different scenarios. At most one of the scenarios will allow to 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 this 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_6_paNr1.zip` or `assignment_6_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 #6 are due to the next tutorial on July 19th, 11:00 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 6.1: Multi-target CMake Setup

Files: `jarvis_algorithm.cpp`, `line_segment_intersection.cpp`

The framework contains two self-contained source code units from *The Algorithms – C++*[2] open source project. These two files are buildable by themselves but do not come with a more sophisticated project setup.

Refactor the given source code to a multi-target project with different kinds of libraries and executables. The goal is to provide the following setup using cmake:

- One header-only library for the geometry classes (a common `Point` class),
- One statically-built or dynamically-built library for the two algorithms *Jarvis* and *Line Segment Intersection* (depends on a CMake Option),
- Two command-line executables for the current test cases.

Further Constraints:

- The project setup is implemented using CMake and should allow for out-of-source builds.
- The project is cross-compilable on the major platforms, i.e., no compiler-specific flags and features are used or they're appropriately selected based on the used compiler.
- The two libraries (geometry and algorithms) use different, non-global namespaces.
- The C++ Standard should be C++11.
- The public API headers are generated through CMake and used appropriately in the algorithms library.
- The default visibility of the libraries is set to hidden to unify link behavior across platforms.

Artifacts:

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

Objectives: cmake setup, multi-target project, library interfaces

Challenge: Add deployment and provisioning for your solution[3]. This includes, that your geometry and algorithm libraries are properly exported and usable in external projects. You may use either Windows Package Manager, Ubuntu PPA, or Homebrew, potentially using personal distribution repositories or server. If you solve this challenge and provide us with the package name and installation process via e-mail to willy.scheibel@hpi.de until July 16th, we'll integrate presentation and discussion of this deployment in the upcoming tutorial.

Task 6.2: Cross-Compilation for Web Applications

Files: `index.html`, `process.cpp`, `start_server.sh`

We want to provide a legacy library with domain-specific code in our web application. The main feature is the computation of the passed argument to the square:

```
1 int process(int n) {
2     return n*n;
3 }
```

This C/C++ functionality should be accessible in browser JavaScript context by using emscripten. Access and usage may be implemented as follows:

```
1 <script src="build/process.js"></script>
2 <script>
3   var process = Module.cwrap('process', 'number', '[number]');
4
5   for (var i = 0; i <= 100; ++i) {
6     console.log("Call with", i, "results in", process(i));
7   }
8 </script>
```

[2] github.com/TheAlgorithms/C-Plus-Plus/
[3] The cmake project template *cmake-init* (github.com/cginternals/cmake-init) is designed for these use cases.

Adapt the given framework by an emscripten build of the C/C++ source code such that the given HTML file uses the cross-compiled source code. We encourage the use of CMake for your solution. The `start_server.sh` can be used to start a local server to test your project.

Artifacts:

a) `...` : Project setup of your solution

b) `build.sh/build.bat`: The script to setup and build your project

Objectives: web deployment, cross-compilation

Task 6.3: Script Embedding in C++

We want to allow to provide functionality in our C++ projects by external scripting, e.g., through JavaScript or Python. Extend the framework to use an external `process` function that provides domain-specific computation.

Artifacts:

a) `CMakelists.txt` : The project setup.

b) `main.cpp` : The extended source code.

Objectives: script embedding

This task allows to choose from multiple scenarios, where at most one may result in a bonus point for the exam. You have to choose between

a) JavaScript integration in C++ with duktape, or

b) Python integration in C++ with the CPython library.

a) JavaScript Integration in C++ with duktape

Files: `duktape/*`, `CMakeLists.txt`, `main.cpp`, `process.js`

The duktape library is provided and is expected to be used as a source code module, i.e., it is build with your build. Your solution should provide setup, usage, and cleanup of the duktape library and state.

b) Python Integration in C++ with the CPython library

Files: `CMakeLists.txt`, `main.cpp`, `process.py`

Python is usable as a library to manage and access Python objects. You may need to install the Python development package on your system. Rely on CMake functionality to integrate Python into your build. Your solution should provide setup, usage, and cleanup of the Python library and state.

Task 6.4: C++ Embedding in Scripts

In contrast to script embedding, we also want to provide C++ functionality in script environments such as Python or JavaScript. This is especially useful to wrap core functionality with high computation demands and make it accessible to higher-level scripting. Extend the framework to use the C/C++ `process` function that provides domain-specific computation in a higher-level scripting context.

Artifacts:

a) `...` : The project setup and source code.

Objectives: C++ embedding in scripts

This task allows to choose from multiple scenarios, where at most one may result in a bonus point for the exam. You have to choose between

a) C++ integration in JavaScript using node.js Modules,

b) C++ integration in Python using cffi.

a) C++ Integration in JavaScript using node.js Modules

Files: `CMakeLists.txt`, `Findnodejs.cmake`, `main.js`, `process.cpp`

Node.js uses the JavaScript engine V8 from Google. You may need to install the node.js development package and adjust the `Findnodejs.cmake` script to integrate the library into your build. If node.js is found, the `NODEJS_INCLUDE_DIRS` CMake variable contains the requires include directories to include in your build. Additional linkage is not required as node.js modules are loaded as a plugin.

Executing the `main.js` file should result in a repeated call to the C/C++ module.

b) C++ Integration in Python using cffi

Files: `main.py`, `process.cpp`

Python provides the `cffi` module to define and pre-compile native C/C++ modules for subsequent usage. Extend the framework such that the functionality is pre-build and provided as module in extern python scripts such as `main.py`

Executing the `main.py` file should result in a repeated call to the C/C++ module.

Additional Artifacts:

a) `build.sh/build.bat`: The script to build the native module