Who includes me?

Getting an #include-s graph with a Clang Plugin

Introduction

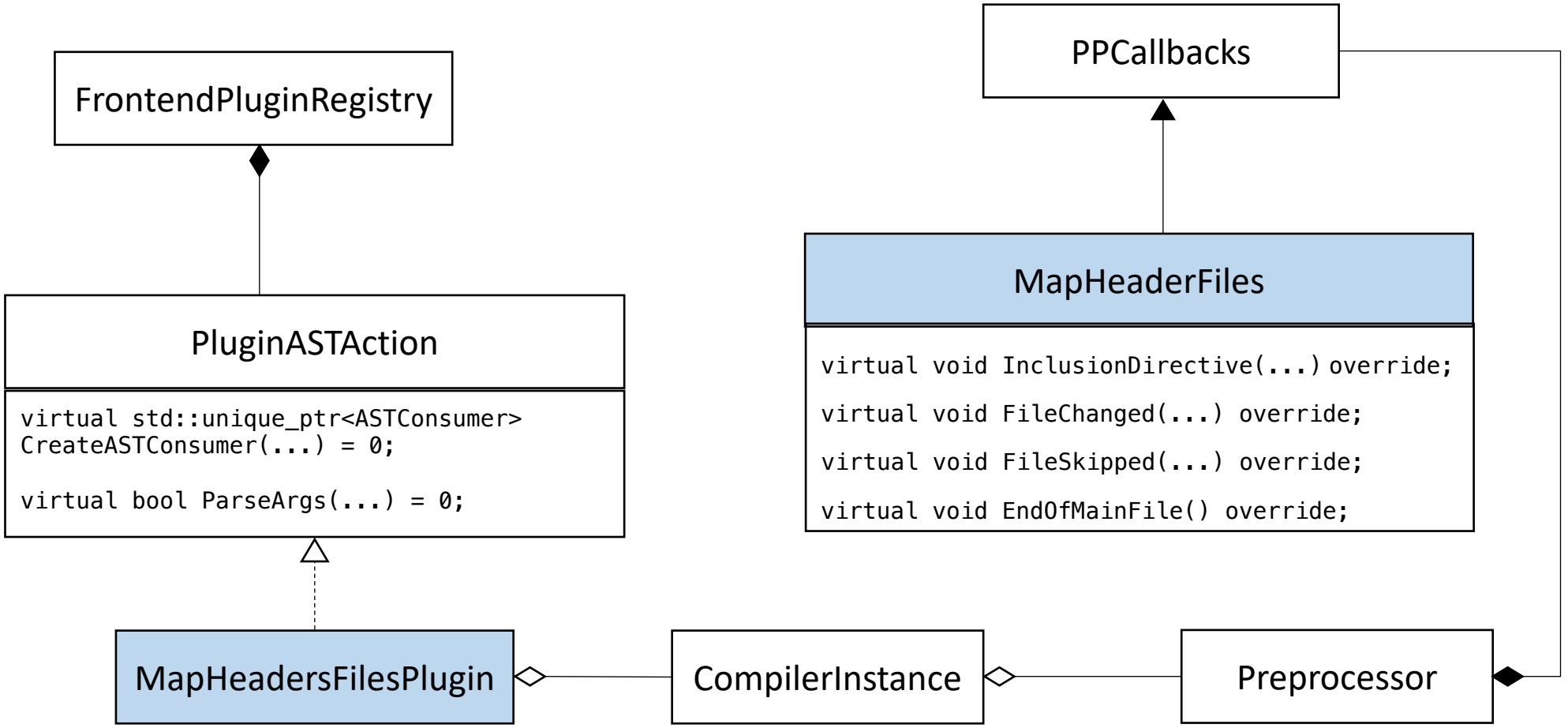
- **WHAT?**
  Create a map of files dependencies in a project, by tracking and registering the trail of header files inclusions
- **WHY?**
  - When dealing with large projects, this information can be interesting, as it reveals the project's design
  - It can also help in identifying redundant or bidirectional inclusions
  - It can assist in evaluating the scope of the effects that a change made in a header file will have on the project
- **HOW?**
  A compiler plugin, that will give us hooks to the compiler's work, in real-time

Plugin's Outline

Usually, a clang plugin implementation will include the following 3 parts:

1. The actual plugin – implementing the **`PluginASTAction`** interface
2. Events listener – overriding callback functions for the compilers events
   - usually an **`ASTConsumer`** that implements logic that concerns the AST*
   - in this case, the logic of the plugin will reside in a class implementing the interface **`PPCallbacks`**
3. Registration of the plugin to Clang's **`FrontendPluginRegistry`**

* Abstract Syntax Tree


FrontendPluginRegistry
PPCallbacks
PluginASTAction
virtual std::unique_ptr<ASTConsumer>
CreateASTConsumer(...) = 0;
virtual bool ParseArgs(...) = 0;
MapHeaderFiles
virtual void InclusionDirective(...) override;
virtual void FileChanged(...) override;
virtual void FileSkipped(...) override;
virtual void EndOfMainFile() override;
MapHeadersFilesPlugin
CompilerInstance
Preprocessor

The listener (type of `PPCallbacks`)

In the `PPCallbacks` implementation, we override the following methods:

```
virtual void FileChanged(...)

virtual void FileSkipped(...)

virtual void InclusionDirective(...)

virtual void EndOfMainFile()
```

I. Stacking header files -

```
virtual void FileChanged(SourceLocation Loc, FileChangeReason Reason,
SrcMgr::CharacteristicKind FileType,  FileID PrevFID = FileID())
```

- This callback is invoked whenever a source file is *entered* or *exited* (=Reason)
- We stack the file names as the compiler enters them recursively – every entered header file is included by the file that is in the top of the stack
  - On entering file, we push to stack
  - On exiting file, we pop from stack

II. Handling skipped files -

```
virtual void FileSkipped(const FileEntry &SkippedFile,
const Token &FilenameTok, SrcMgr::CharacteristicKind FileType)
```

- Header files are often skipped, for optimization - the compiler will not enter them, so the callback defined above will not be invoked
- This defeats the purpose of creating an exhaustive #include-s map
- In order to stack these headers too, we will override this method too

III. Handling system headers -

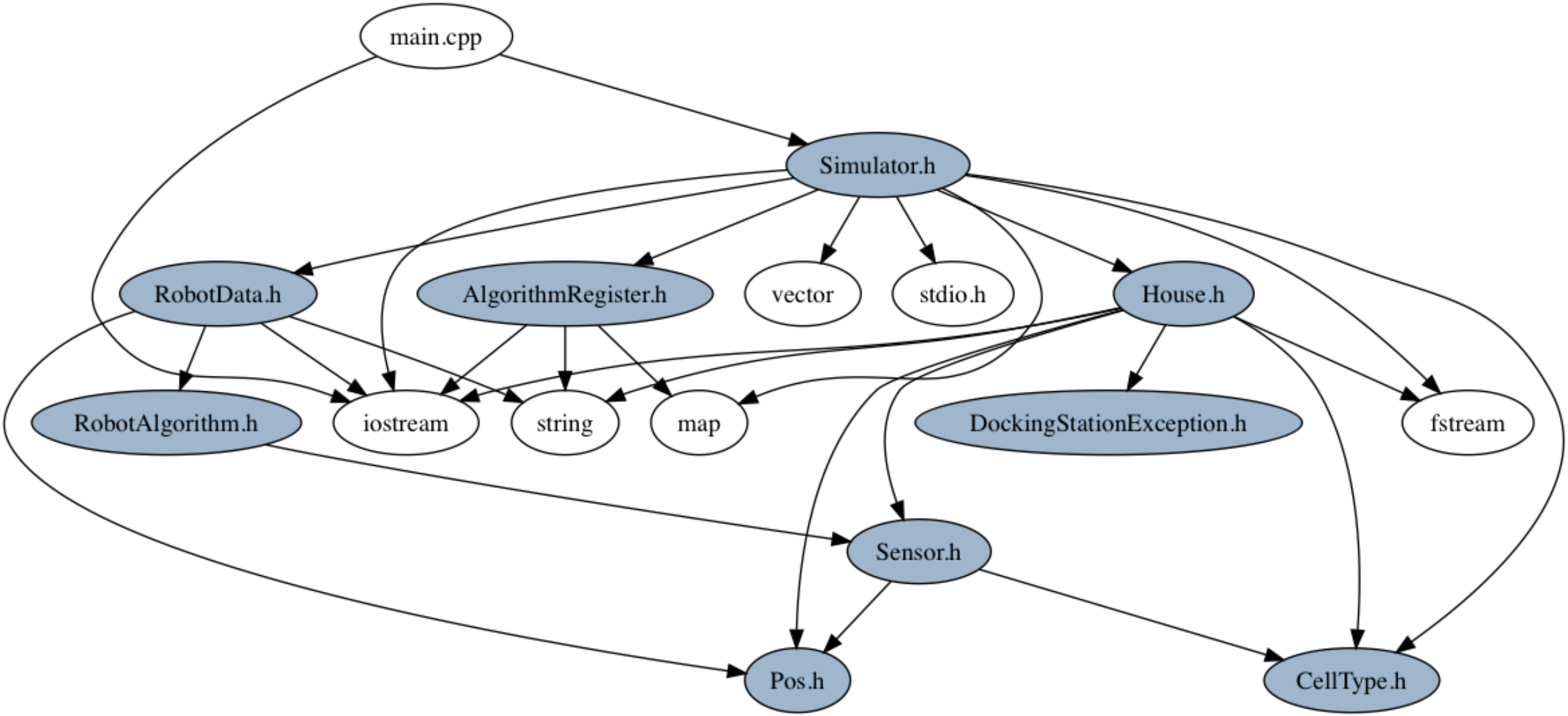
```
  virtual void InclusionDirective(SourceLocation HashLoc, const Token
&IncludeTok, StringRef FileName, bool IsAngled, CharSourceRange
FilenameRange, const FileEntry *File, StringRef SearchPath, StringRef
RelativePath, const Module *Imported, SrcMgr::CharacteristicKind FileType)
```

- To avoid (overwhelming) clutter, we can narrow our mapping to header files that are created by the user
  - this can be decided by an argument to the plugin
- In order to exclude system headers from the full depth headers mapping, we need to identify them
- These headers are usually identified by the < > delimiters – which are present in the actual inclusion directive

IV. Saving the finding -

```
virtual void EndOfMainFile()
```

- We will write to a file all of the header names we've collected
- This information will be saved in a format that expresses the hierarchy between the files: the *included* files are children of the *including* file
- To get this graph visually, we can write this information in the format required by **Graphviz**


main.cpp
Simulator.h
RobotData.h
AlgorithmRegister.h
vector
stdio.h
House.h
RobotAlgorithm.h
iostream
string
map
DockingStationException.h
fstream
Sensor.h
Pos.h
CellType.h

Standing on the shoulders of giants