INSTITUTO TECNOLOGICO DE IZTAPALAPA

Engineering in computer systems

Clang: a C language family frontend for LLVM

It presents:
STUDENT NAME
Alvarez Rosales Alejandro
Ramírez Peña Carlos Iván
Rodríguez Pérez Luis Diego
Hernández Ruiz Adrián Felipe

Control number:
151080010
161080215
161080170
171080105

Internal Advisor:
PARRA HERNANDEZ ABIEL TOMAS

MEXICO CITY **11 / 20**

Project Clang front end

Clang Features

One of the greatest advantages of Clang is the modular design of its development, which seems to be one of the pillars from the conception of the project. This allows us to have a very well-designed API and to be able to use it to build our own tools for code analysis, in the same way that the front-end does when compiling it. In other words, as an open-source project, we can reuse the libraries that the Clang project offers us an easily embed the compiler in the new applications that we are developing, which is not so simple in other compilers like GCC. Among the activities that we can undertake thanks to this, we can mention static analysis, refactoring or code generation.

Here are some of the benefits that the Clang project proclaims:

- One of its objectives is to reduce compilation time and memory usage. Its architecture can allow more directly profiling the cost of each layer. Also, the less memory the code takes, the more of it can be put into memory at the same time, which benefits code analysis.
- Report errors and warnings in a very expressive way so that it is as useful as possible, indicating exactly the point at which the error occurs and information related to it.

- Clang seeks that the developed tools can be easily integrated with Integrated Development Environments (IDEs), in order to expand their scope of application.

- It uses the BSD license, which allows Clang to be used in commercial products.

- Clang tries to conform as completely as possible to the standards of the C family languages and their variants. For those supported extensions that somehow do not officially agree with the standard, they are issued as notices for the developer's knowledge.

On the other hand, we can comment on the following advantages of Clang over GCC:

- Regarding compilation speed and memory usage, it has been verified, independently of Clang, that compile time is typically better than using GCC. However, the runtime is better with GCC, although Clang has made a lot of progress in this regard in recent years, and an even greater improvement is expected thanks to the constant activity in its development.

- The BSD license is more permissive than the GCC GPL, which does not allow embedding the tool in software that is not licensed as the GPL.

- Language standards are more enforced. For example, something like ' void f (int a, int a);' is not allowed.

- When it comes to error and warning messages, Clang provides a lot more information, such as the exact column of the error, the range affected, and even suggestions for improvement.
- Support is included for a greater number of language extensions and inherits useful LLVM features such as a backend (for example: support for binding time optimization)
- Last but not least, Clang has documentation and is supported by an active mailing list where you can find help with problems with the use of the tool. (Delgado Pérez, 2015, 16)

Clang con C++

Clang fully implements all published ISO C ++ standards (C ++ 98 / C ++ 03, C ++ 11, C ++ 14, and C ++ 17), and some of the upcoming C ++ 20 standards.

The Clang community continually strives to improve C ++ standards compliance between versions by submitting and tracking C ++ defect reports and implementing solutions as they become available.

Experimental work is also underway to implement C ++ technical specifications that will help drive the future of the C ++ programming language.

The LLVM bug tracker contains Clang C ++ components that track known bugs with Clang language compliance in each language mode.

C ++ 98 deployment status

Clang implements the entire ISO C ++ 1998 standard (including the flaws addressed in the ISO C ++ 2003 standard) except for export (which was removed in C ++ 11).

C ++ 11 deployment status

Clang 3.3 and later versions implement the entire ISO C ++ 2011 standard.
By default, Clang creates C ++ code according to the C ++ 98 standard, with many C ++ 11 features accepted as extensions. You can use Clang in C ++ 11 mode with the -std = c ++ 11 option. Clang's C ++ 11 mode can be used with either libc ++ or gcc's libstdc ++.

C ++ 14 deployment status

Clang 3.4 and later versions implement the entire ISO C ++ 2014 standard.
You can use Clang in C ++ 14 mode with the -std = c ++ 14 option (use -std = c ++ 1 and in Clang 3.4 and earlier).

C ++ 17 deployment status

Clang 5 and later versions implement all the features of the ISO C ++ 2017 standard. You can use Clang in C ++ with 17 mode - std = c ++ 17 option (use - std = c ++ 1zen Clang 4 and above).

C ++ 20 deployment status

Clang has support for some of the features of the ISO C ++ 2020 Draft International Standard. You can use Clang in C ++ with 20 mode - std = c ++ 20 option (use - std = c ++ 2aen Clang 9 and above). (LLVM, nd.)

Project tools

Currently, clang is divided into the following libraries and tools:

- **libsupport:** Basic support library, from LLVM.
- **libsystem** : system abstraction library, from LLVM.
- **libbasic** - diagnostics, source locations, Source Buffer abstraction , file system caching for input source files.
- **libast** : provides classes to represent CAST, the C-type system, built-in functions, and various helpers to parse and manipulate the AST (visitors, pretty printers, etc.).
- **liblex** : Lexing and preprocessing, identifier hash table, pragma handling, tokens and macro expansion.
- **get rid** : analysis. This library invokes detailed 'Actions' provided by the client (Shample libsema builds AST) but does not know anything about AST or other client specific data structures.
- **libsema** - Semantic analysis. This provides a set of parser actions to build a standardized AST for programs.
- **libcodegen** : **lower** the AST to LLVM IR for optimization and code generation.
- **librewrite** : editing text buffers (important for code rewriting transformation, such as refactoring).
- **libanalysis** - Static analysis support.
- **clang** - A driver program, client of libraries at various levels.

Posible problems

Analysis on the go

CLion constantly monitors your code for possible errors. If it finds something, it highlights the suspicious code snippet in the editor. If you look at the edit gutter on the right, you will see yellow and red error strips which, if clicked, will take you to the detected problems. Another way to navigate from one highlighted issue to another is by pressing F2 / Shift + F2 . The status indicator at the top of the gutter summarizes the status of the file.
In addition to finding compilation errors, CLion identifies inefficiencies in your code and even performs a data flow analysis of your code, to locate inaccessible / unused code, as well as other problems and "code stinks."

Quick fixes

CLion's on-the-fly code inspections cover about 40 potential problem cases in C / C ++ code and a few in other languages as well.
When a problem is highlighted, hover over it, press Alt + Enter, and choose from suggested quick fixes. (You can also access the context menu by clicking the light bulb next to the line.)
You can also choose to fix all the similar problems in your project. Or, if you don't find this inspection helpful, you can tailor it to suit your needs.

Inspect the code

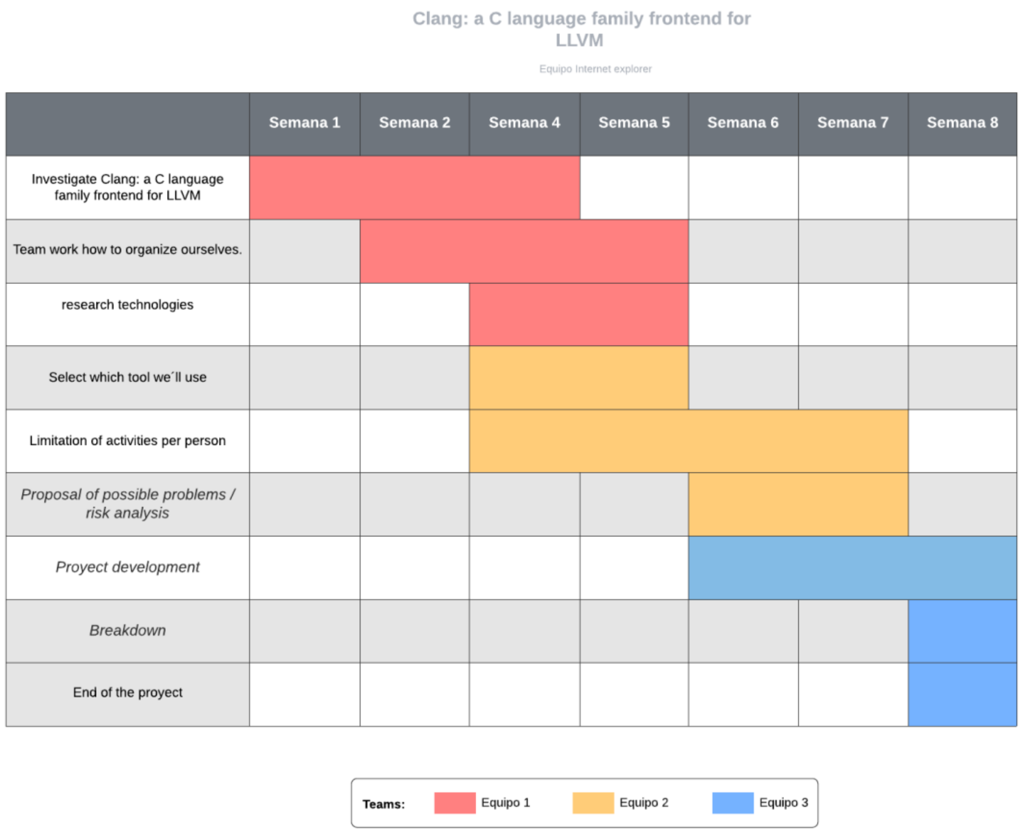
CLion provides detailed descriptions of all available inspections. You can also manage its scope (choose from Typo , Warning , Error, etc.) or even, in some cases, adjust the parameters of an inspection to better reflect your requirements.
You can run multiple (or even all) inspections in this batch mode with Code | Inspect Code .
If you want to remove a particular problem from your entire code base, you can use Run Watch by Name (Ctrl + Alt + Shift + I) and select the desired scope. A separate window will open with the inspection results, where you can regroup problems and apply quick batch fixes to all problems, when possible.

There are several inspections implemented integrated into the engine based on Clangd personalized CLION :

- Member function can be static
- Argument selection errors
- Statement or empty statement
- Constructor or destructor virtual call
- Unused includes

The " unused includes " check suggests 3 detection strategies: one conservative, one aggressive, and one default (*Detect not directly used*), which is the closest to the beginning " Include What You Use".

Clang-Tidy
CLion comes with the Clang-Tidy integration . Clang-Tidy checks are displayed in the same way as CLion's built-in code inspections , and it also provides quick fixes via Alt + Enter .
Go to Settings / Preferences | Editor | Inspections | C / C ++ | General | Clang-Tidy to adjust the list of enabled / disabled checks in CLion . The Clang-Tidy command line format is used in the text field. You can see the default settings. Or use the configuration files . *clang - tidy* instead of the settings provided by the IDE.
Also, individual checks can be enabled / disabled via a context menu.
Enable C ++ Core Guidelines or Clang Static Analyzer checks , try update checks or even implement your own checks and receive them immediately in CLion (for custom checks, change the Clang-Tidy binary to your own in Settings / Preferences | Languages & Frameworks | C / C ++).



Clang: a C language family frontend for LLVM

Equipo Internet explorer

	Semana 1	Semana 2	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8
Investigate Clang: a C language family frontend for LLVM	🟥	🟥	🟥				
Team work how to organize ourselves.		🟥	🟥	🟥			
research technologies			🟥	🟥			
Select which tool we´ll use			🟨	🟨			
Limitation of activities per person			🟨	🟨	🟨	🟨	
Proposal of possible problems / risk analysis					🟨	🟨	
Proyect development					🟦	🟦	🟦
Breakdown							🟦
End of the proyect							🟦

Teams: 🟥 Equipo 1 🟨 Equipo 2 🟦 Equipo 3

Schedule of activities