Lenguajes y Autómatas II

Nombre del equipo: INTERNET EXPLORER 95

Nombre de integrante: Rodríguez Perez Luis diego
no control: 161080170

nombre de integrante: Ramírez Peña Carlos Iván
no control: 161080215

nombre de integrante: Hernández Ruiz Adrián Felipe
no control:171080105

nombre de integrante: Álvarez Rosales Alejandro
no control: 151080010

Grupo: 7 AM

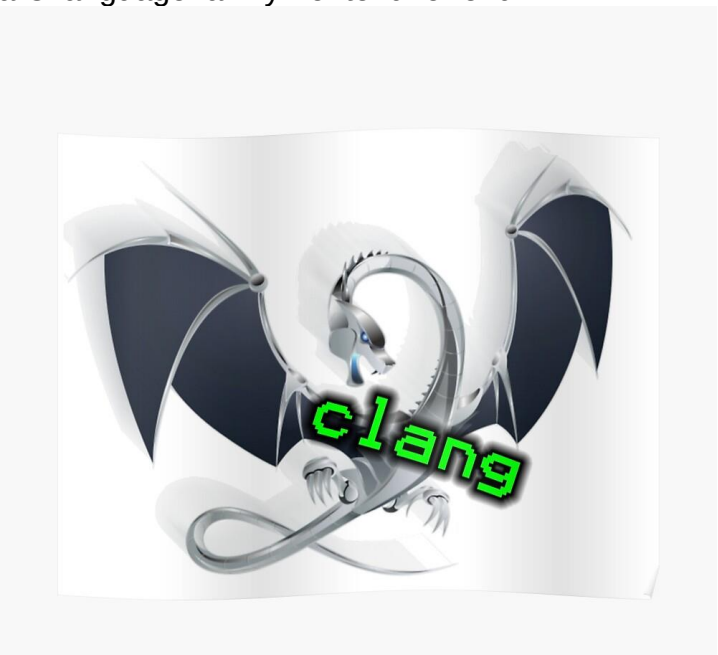
NOMBRE DE LA ESCUELA: Instituto Tecnológico de Iztapalapa

Carrera: Ing. en Sistemas computacionales

Nombre de la materia: Lenguajes y Autómatas 2

Nombre del Profesor: Parra Hernández Abiel

tema: proyecto. - Clang: a C language family frontend for end



Proyecto Clang front end

Características de Clang

Una de las mayores ventajas de Clang es el diseño modular de su desarrollo, que parece ser uno de los pilares desde la concepción del proyecto. Esto nos permite disponer de una API muy bien diseñada y poder emplearla para la construcción de nuestras propias herramientas para el análisis del código, de la misma manera en la que el front-end lo hace al compilarlo. En otras palabras, como proyecto de código abierto, podemos reutilizar las bibliotecas que nos ofrece el proyecto Clang para embeber de forma sencilla el compilador en las nuevas aplicaciones que estemos desarrollando, lo cual no es tan sencillo en otros compiladores como GCC. Entre las actividades que podemos emprender gracias a esto, podemos mencionar el análisis estático, la refactorización o la generación de código.

A continuación, se enumeran algunas de las ventajas que el proyecto Clang proclama:

- Uno de sus objetivos es reducir el tiempo de compilación y el uso de la memoria. Su arquitectura puede permitir de forma más directa el crear perfiles del coste de cada capa. Además, cuanta menos memoria toma el código, mayor cantidad del mismo se podrá incluir en memoria al mismo tiempo, lo cual beneficia el análisis del código.

- Informa de errores y avisos de una manera muy expresiva para que sea lo más útil posible, indicando exactamente el punto en el que se produce el error e información relacionada con el mismo.

- Clang busca que las herramientas desarrolladas puedan integrarse fácilmente con Entornos de Desarrollo Integrados (IDEs), a fin de que su ámbito de aplicación se amplíe.

- Utiliza la licencia BSD, la cual permite que Clang sea empleado en productos comerciales.

- Clang trata de ajustarse de la forma más completa posible a los estándares de los lenguajes de la familia C y sus variantes. Para aquellas extensiones soportadas que, de alguna manera no concuerdan de manera oficial con el estándar, son emitidas como avisos para conocimiento del desarrollador.

Por otra parte, podemos comentar las siguientes ventajas de Clang sobre GCC:

- En cuanto a la velocidad de compilación y uso de memoria, se ha verificado, de forma independiente a Clang, que el tiempo de compilación es normalmente mejor que usando GCC. Sin embargo, el tiempo de ejecución es mejor con GCC, aunque Clang ha medrado mucho en este aspecto en los últimos años, y se espera una mejora aún mayor gracias a la actividad constante en su desarrollo.

- La licencia BSD es más permisiva que la GPL de GCC, lo cual no permite embeber la herramienta en software que no esté licenciado como GPL.

- Se hacen cumplir más las normas de los lenguajes. Por ejemplo, no se permite algo como 'void f(int a, int a);'.

- En cuanto a los mensajes de error y aviso, Clang proporciona bastante más información, como la columna exacta del error, el rango afectado e incluso sugerencias de mejora.

- Se incluye soporte para un mayor número de extensiones del lenguaje y hereda características útiles de LLVM como backend (por ejemplo, soporte para la optimización del tiempo de enlazado).

- Por último, y no menos importante, Clang dispone de documentación y tiene soporte mediante una lista de correos activa en la que se puede encontrar ayuda a los problemas con el uso de la herramienta. (Delgado Pérez, 2015, 16)

Clang con C++

Clang implementa completamente todos los estándares ISO C ++ publicados (C ++ 98 / C ++ 03 , C ++ 11 , C ++ 14 y C ++ 17), y algunos de los próximos estándares C ++ 20.
La comunidad de Clang se esfuerza continuamente por mejorar el cumplimiento de los estándares de C ++ entre versiones mediante el envío y seguimiento de informes de defectos de C ++ e implementando soluciones a medida que están disponibles.
También se están realizando trabajos experimentales para implementar las especificaciones técnicas de C ++ que ayudarán a impulsar el futuro del lenguaje de programación C ++.
El rastreador de errores de LLVM contiene componentes de Clang C ++ que rastrean los errores conocidos con la conformidad del lenguaje de Clang en cada modo de idioma.

Estado de implementación de C ++ 98

Clang implementa todo el estándar ISO C ++ 1998 (incluidos los defectos abordados en el estándar ISO C ++ 2003) excepto para la exportación (que se eliminó en C ++ 11).

Estado de implementación de C ++ 11

Clang 3.3 y versiones posteriores implementan todo el estándar ISO C ++ 2011.
De forma predeterminada, Clang crea código C ++ de acuerdo con el estándar C ++ 98, con muchas características de C ++ 11 aceptadas como extensiones. Puede usar Clang en modo C ++ 11 con la -std=c++11 opción. El modo C ++ 11 de Clang se puede usar con libc ++ o con libstdc ++ de gcc.

Estado de implementación de C ++ 14

Clang 3.4 y versiones posteriores implementan todo el estándar ISO C ++ 2014.
Puede usar Clang en modo C ++ 14 con la -std=c++14 opción (usar -std=c++1 yen Clang 3.4 y anteriores).

Estado de implementación de C ++ 17

Clang 5 y versiones posteriores implementan todas las características del estándar ISO C ++ 2017.
Puede usar Clang en modo C ++ 17 con la -std=c++17 opción (usar -std=c++1zen Clang 4 y anteriores).

Estado de implementación de C ++ 20

Clang tiene soporte para algunas de las características del Borrador de Norma Internacional ISO C ++ 2020.
Puede usar Clang en modo C ++ 20 con la -std=c++20 opción (usar -std=c++2aen Clang 9 y anteriores). (LLVM, n.d.)

Herramientas del proyecto

Actualmente, clang se divide en las siguientes bibliotecas y herramientas:

- **libsupport** : biblioteca de soporte básico, de LLVM.
- **libsystem** : biblioteca de abstracción del sistema, de LLVM.
- **libbasic** : diagnósticos, ubicaciones de origen, abstracción de Source Buffer, almacenamiento en caché del sistema de archivos para archivos de origen de entrada.
- **libast** : proporciona clases para representar CAST, el sistema de tipo C, funciones integradas y varios ayudantes para analizar y manipular el AST (visitantes, impresoras bonitas, etc.).
- **liblex** : Lexing y preprocesamiento, tabla hash de identificadores, manejo de pragma, tokens y expansión de macros.

- **librarse** : análisis. Esta biblioteca invoca 'Acciones' detalladas proporcionadas por el cliente (por ejemplo, libsema construye AST) pero no sabe nada sobre AST u otras estructuras de datos específicas del cliente.

- **libsema** - Análisis semántico. Esto proporciona un conjunto de acciones del analizador para construir un AST estandarizado para programas.

- **libcodegen** : **baje** el AST a LLVM IR para optimización y generación de código.

- **librewrite** : edición de búferes de texto (importante para la transformación de reescritura de código, como la refactorización).

- **libanalysis** - Soporte de análisis estático.

- **clang** - Un programa controlador, cliente de las bibliotecas en varios niveles.

Posibles problemas

Análisis sobre la marcha

CLion supervisa constantemente su código para detectar posibles errores. Si encuentra algo, resalta el fragmento de código sospechoso en el editor. Si usted observa el medianil de edición de la derecha, verá tiras de errores amarillas y rojas que, si se hace clic en ellas, lo llevarán a los problemas detectados. Otra forma de navegar de una incidencia resaltada a otra es pulsando F2/Shift+F2. El indicador de estado en la parte superior del medianil resume el estado del archivo.
Además de encontrar errores de compilación, CLion identifica las ineficiencias del código e incluso realiza un análisis de flujo de datos de su código, para ubicar el código inaccesible / no utilizado, así como otros problemas y «hediondeces del código».

Arreglos rápidos

Las inspecciones de código sobre la marcha de CLion cubren alrededor de 40 casos de problemas potenciales en el código C/C ++ y también unos cuantos en otros lenguajes.
Cuando se resalta un problema, coloque el cursor en él, pulse Alt+Enter y elija entre las soluciones rápidas sugeridas. (También puede acceder al menú contextual haciendo clic en la bombilla junto a la línea.)
También puede optar por solucionar todos los problemas similares en su proyecto. O, si no encuentra útil esta inspección, la puede adaptar a sus necesidades.

Inspeccionar el código

CLion proporciona descripciones detalladas de todas las inspecciones disponibles. También puede gestionar su alcance (elija entre Typo, Warning, Error, etc.) o incluso, en algunos casos, ajustar los parámetros de una inspección para que refleje mejor sus requisitos.
Puede ejecutar varias inspecciones (o incluso todas) en este modo por lotes con Code | Inspect Code.
Si desea eliminar un problema en particular de toda su base de código, puede utilizar Ejecutar inspección por nombre (Ctrl+Alt+Shift+I) y seleccionar el alcance deseado. Se abrirá una ventana separada con los resultados de la inspección, en la que puede reagrupar los problemas y aplicar soluciones rápidas por lotes a todos los problemas, cuando sea posible.

Hay varias inspecciones implementadas integradas en el motor basado en Clangd personalizado de CLion:

- La función de miembro puede ser estática
- Errores de selección de argumento
- Instrucción o declaración vacía
- Llamada virtual de constructor o destructor
- Unused includes

La comprobación «unused includes» sugiere 3 estrategias de detección: una conservadora, una agresiva y una predeterminada (*Detect not directly used*), qué es la más parecida al principio «Include What You Use».

Clang-Tidy

CLion viene con la integración de Clang-Tidy. Las verificaciones de Clang-Tidy se muestran de la misma manera que las inspecciones de código incorporadas de CLion, y también proporciona soluciones rápidas a través de Alt+Enter.

Ir a Settings/Preferences | Editor | Inspections | C/C++ | General | Clang-Tidy para ajustar la lista de verificaciones habilitadas / deshabilitadas en CLion. El formato de la línea de comando de Clang-Tidy se utiliza en el campo de texto. Puede ver la configuración predeterminada. O utilizar los archivos de configuración *.clang-tidy* en lugar de la configuración proporcionada por el IDE.
Además, las verificaciones individuales se pueden habilitar / deshabilitar a través de un menú contextual.
Habilite C++ Core Guidelines o verificaciones de Clang Static Analyzer, pruebe actualización de verificaciones o incluso implemente sus propias verificaciones y recíbelas inmediatamente en CLion (para verificaciones personalizadas, cambie el binario de Clang-Tidy al suyo propio en Settings/Preferences | Languages & Frameworks | C/C++).