INSTITUTO TECNOLÓGICO DE IZTAPALAPA

Ingeniería en Sistemas Computacionales

Clang: a C language family frontend for LLVM

Presenta:
NOMBRE DEL ESTUDIANTE
Álvarez Rosales Alejandro
Ramírez Peña Carlos Iván
Rodríguez Pérez Luis Diego
Hernández Ruiz Adrián Felipe

No. De Control:
151080010
161080215
161080170
171080105

Asesor Interno:
PARRA HERNANDEZ ABIEL TOMAS

CIUDAD DE MÉXICO **11 / 20**

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++).

FECHA DE INICIO:	30/11/2020							FECHA DE TERMINACIÓN	24/01/2021
OBJETIVO DEL PROYECTO:	Conocer el front end del compilador con clang								
ACTIVIDAD		1	2	3	4	5	6	7	8
Investigar Clang: a C language family frontend for LLVM	P								
	R								
Trabajos en equipo cómo organizarnos.	P								
	R								
Ivestigar las tecnologías	P								
	R								
Selección de herramientas	P								
	R								
Delimitación de actividades por persona	P								
	R								
Planteamiento de posibles escenarios/analisis de riesgos	P								
	R								
Desarrollo del proyecto	P								
	R								
Desgloce	P								
	R								
Finalización del proyecto	P								
	R								
OBSERVACIONES:									

Análisis de riesgo.

Analisis de riesgo					
RIESGO	SOLUCIÓN	BAJO	MODERADO	ALTO	MUY ALTO
compatibilidad con el editor	usar un editor adecuado			X	
errores de codigo	analizar el codigo		X		
extenciones no conocidas	buscar las extenciones necesarias		X		
librerias	buscas la liberias necesarias conforme al avance			X	
problemas con la estabilidad del editor	buscar la solucion en los archivos .json			X	
saber identificar el problema	saber que hace cada linea de codigo				X
versiones con las herraminetas a utilizar	hacer una investigacion, para usar la mas adecuada			X	
Familiarizarse con el lenguaje	hacer ejecicios del lenguaje				X
problemas con la terminal de el editor	configuración del editor		X		
vinculación con clang	configuración del editor		X		

Descripción de la infraestructura Clang

El proyecto Clang proporciona una infraestructura de herramientas y front-end de lenguaje para lenguajes de la familia de lenguajes C (C, C ++, Objective C / C ++, OpenCL, CUDA y RenderScript) para el proyecto LLVM. Se proporcionan tanto un controlador de compilador compatible con GCC (clang) como un controlador de compilador compatible con MSVC (clang-cl.exe).

Funciones y objetivos:

Características del usuario final:

- Compilaciones rápidas y poco uso de memoria
- Diagnóstico expresivo
- Compatibilidad con GCC

Utilidad y aplicaciones:

- Arquitectura modular basada en bibliotecas
- Soporte a diversos clientes (refactorización, análisis estático, generación de código, etc.)
- Permitir una estrecha integración con los IDE
- Utilice la licencia LLVM 'Apache 2'

Diseño e implementación internos:

- Un compilador de calidad de producción del mundo real
- Una base de código simple y pirateable
- Un único analizador unificado para C, Objective C, C ++ y Objective C ++
- Conformidad con C / C ++ / ObjC y sus variantes

¿Por qué?

El desarrollo del nuevo front-end se inició debido a la necesidad de un compilador que permita mejores diagnósticos, mejor integración con IDE, una licencia que sea compatible con productos comerciales y un compilador ágil que sea fácil de desarrollar y mantener. Todos estos fueron motivos para comenzar a trabajar en una nueva interfaz que pudiera satisfacer estas necesidades.

Estado actual

Clang se considera un compilador C, Objective-C, C ++ y Objective-C ++ de calidad de producción cuando se dirige a X86-32, X86-64 y ARM (otros destinos pueden tener salvedades, pero generalmente son fáciles de solucionar). Por ejemplo, Clang se utiliza en producción para crear software de rendimiento crítico como Chrome o Firefox. Clang es compatible con C ++ 11, C ++ 14 y C ++ 17.

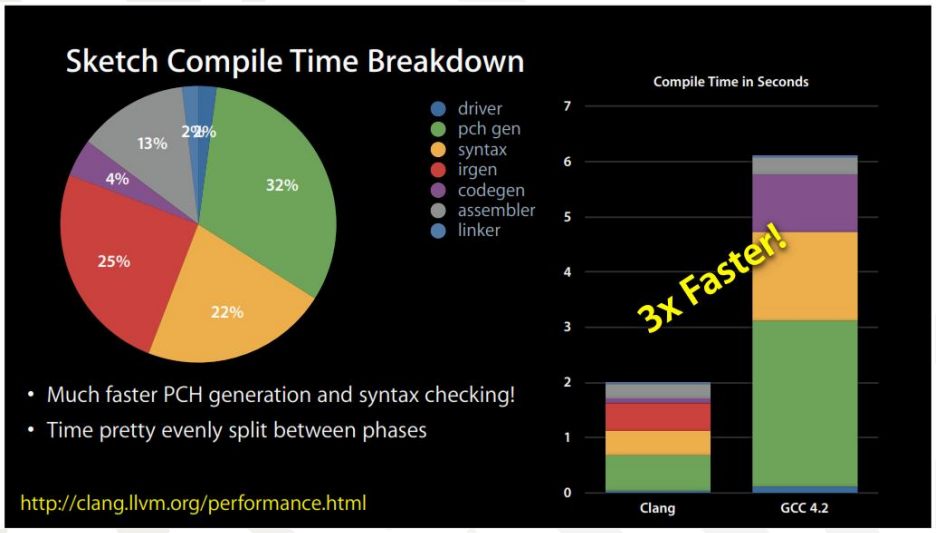
Actuación del Front-end Clang:

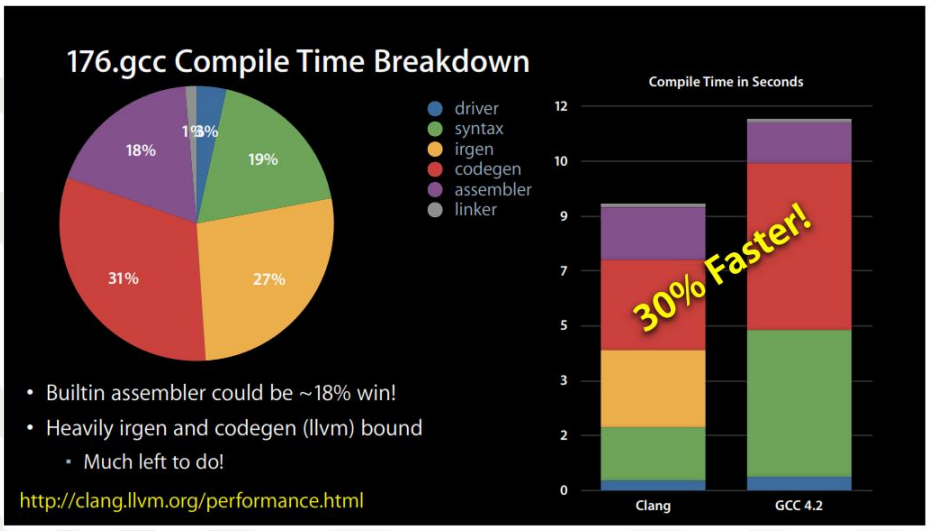
C y C++ son hostiles a los tiempos de compilación rápidos:

- Muchas fases de traducción: trigrafos, líneas nuevas escapadas, macros
- El texto #incluye los encabezados (grandes)
- Abuso del sistema de archivos buscando los archivos de cabecera

Tanto los usuarios como las herramientas de compilación quieren compilaciones rápidas:

- Comparamos los tiempos de compilación del depurador "-O0 -g" vs GCC 4.2





Justificación

El LLVM tiene cubierto para todos los idiomas calientes

Ir:

- Toca el poder del lenguaje Go de Google
- Los mejores ID de Go language y editores.

Kotlin:

- ¿Qué es Kotlin? La alternativa de Java explicada.
- Kotlin frameworks: una encuesta de herramientas de desarrollo de JVM

Pitón:

- ¿Qué es Python? Todo lo que necesitas saber
- Tutorial: Cómo empezar con Python
- 6 bibliotecas esenciales para cada desarrollador de Python

Moho:

- ¿Qué es el óxido? La forma de hacer un desarrollo de software seguro, rápido y fácil.
- Aprende cómo comenzar con Rust en el tutorial.

LLVM: diseñado para la portabilidad

Una forma de pensar en LLVM es como algo que se dice a menudo sobre el lenguaje de programación C: C se describe a veces como un lenguaje ensamblador portátil de alto nivel, porque tiene construcciones que se pueden relacionar estrechamente con el hardware del sistema y se ha adaptado. A casi todas las arquitecturas de sistemas las hay. Pero C solo funciona como un lenguaje de ensamblaje portátil como un efecto secundario de cómo funciona; Ese no era realmente uno de sus objetivos de diseño.

Por el contrario, el IR de LLVM se diseñó desde el principio para ser un conjunto portátil. Una forma de lograr esta portabilidad es ofreciendo primitivos independientes de cualquier arquitectura de máquina en particular. Por ejemplo, los tipos de enteros no se limitan al ancho máximo de bits del hardware subyacente (como 32 o 64 bits). Puede crear tipos de enteros primitivos utilizando tantos bits como sea necesario, como un entero de 128 bits. Tampoco tiene que preocuparse por crear una salida para que coincida con el conjunto de instrucciones de un procesador específico; LLVM se encarga de eso también por ti.

Si desea ver ejemplos en vivo de cómo se ve LLVM IR, vaya al sitio web del Proyecto ELLCC y pruebe la demostración en vivo que convierte el código C en LLVM IR directamente en el navegador.

Cómo los lenguajes de programación usan LLVM

El caso de uso más común para LLVM es como un compilador adelantado para un idioma (AOT). Pero LLVM también hace posible otras cosas.

Compilación justo a tiempo con LLVM

Algunas situaciones requieren que el código se genere sobre la marcha en tiempo de ejecución, en lugar de compilarlo antes de tiempo. El lenguaje Julia, por ejemplo, JIT-compila su código, porque necesita ejecutarse rápidamente e interactuar con el usuario a través de un REPL (bucle de lectura-evaluación-impresión) o mensaje interactivo. Mono, la implementación de .Net, tiene una opción para compilar en código nativo a través de un back-end LLVM.

Numba, un paquete de aceleración matemática para Python, JIT-compila las funciones seleccionadas de Python a código de máquina. También puede compilar código decorado con Numba con anticipación, pero (como Julia) Python ofrece un rápido desarrollo al ser un lenguaje interpretado. El uso de la compilación JIT para producir dicho código complementa el flujo de trabajo interactivo de Python mejor que la compilación anticipada.

Otros están experimentando con formas poco ortodoxas de usar LLVM como JIT, como la compilación de consultas de PostgreSQL, lo que produce un aumento de rendimiento hasta cinco veces mayor.

Optimización automática de código con LLVM

LLVM no solo compila el IR a código de máquina nativo. También puede programarlo para que optimice el código con un alto grado de granularidad, durante todo el proceso de vinculación. Las optimizaciones pueden ser bastante agresivas, incluidas cosas como funciones de alineación, eliminación de código muerto (incluidas declaraciones de tipo no utilizadas y argumentos de función), y desenrollar bucles.

LLVM puede manejarlos por usted, o puede dirigirlos para que se desactiven según sea necesario. Por ejemplo, si desea binarios más pequeños a costa de algún rendimiento, puede hacer que el extremo frontal de su compilador le indique a LLVM que deshabilite el desenrollado del bucle.

Lenguajes específicos de dominio con LLVM

LLVM se ha utilizado para producir compiladores para muchos lenguajes de propósito general, pero también es útil para producir lenguajes que son altamente verticales o exclusivos de un dominio de problemas. De alguna manera, aquí es donde LLVM brilla más, ya que elimina gran parte de la monotonía al crear un lenguaje de este tipo y lo hace funcionar bien.

El proyecto Emscripten, por ejemplo, toma el código IR de LLVM y lo convierte en JavaScript, en teoría permite que cualquier idioma con un back-end de LLVM exporte código que pueda ejecutarse en el navegador. El plan a largo plazo es tener servicios de fondo basados en LLVM que puedan producir WebAssembly, pero Emscripten es un buen ejemplo de lo flexible que puede ser LLVM.

Trabajando con LLVM en varios idiomas

La forma típica de trabajar con LLVM es a través del código en un idioma con el que se sienta cómodo (y eso es compatible con las bibliotecas de LLVM, por supuesto).

Dos opciones de lenguaje comunes son C y C +. Muchos desarrolladores de LLVM predeterminan uno de esos dos por varias buenas razones:

- LLVM en sí está escrito en C +.
- Las API de LLVM están disponibles en encarnaciones C y C +.
- Mucho desarrollo del lenguaje tiende a ocurrir con C / C + como base

Aun así, esos dos idiomas no son las únicas opciones. Muchos idiomas pueden llamar de forma nativa a las bibliotecas de C, por lo que teóricamente es posible realizar el desarrollo de LLVM con cualquier lenguaje de este tipo. Pero ayuda tener una biblioteca real en el lenguaje que envuelve elegantemente las API de LLVM. Afortunadamente, muchos idiomas y tiempos de ejecución de idiomas tienen tales bibliotecas, como C # /. Net / Mono, Rust, Haskell, OCAML, Node.js, Go y Python.

Una advertencia es que algunos de los enlaces de lenguaje a LLVM pueden ser menos completos que otros. Con Python, por ejemplo, hay muchas opciones, pero cada una varía en su integridad y utilidad:

- El proyecto LLVM mantiene su propio conjunto de enlaces a la API C de LLVM, pero actualmente no se mantienen.
- llvmpy quedó fuera de mantenimiento en 2015: malo para cualquier proyecto de software, pero más aún cuando se trabaja con LLVM, dado el número de cambios que aparecen en cada edición de LLVM.
- llvmlite, desarrollado por el equipo que crea Numba, se ha convertido en el contendiente actual para trabajar con LLVM en Python.
- llvmcpy tiene como objetivo actualizar los enlaces de Python para la biblioteca C, mantenerlos actualizados de manera automática y hacerlos accesibles utilizando los lenguajes nativos de Python. llvmcpy aún se encuentra en las primeras etapas, pero ya puede hacer un trabajo rudimentario con las API de LLVM.

Si tiene curiosidad acerca de cómo usar las bibliotecas de LLVM para crear un lenguaje, los creadores de LLVM tienen un tutorial, que usa C + o OCAML, que lo guía a través de la creación de un lenguaje simple llamado Kaleidoscope. Desde entonces ha sido portado a otros idiomas:

- **Haskell:** Un puerto directo del tutorial original.
- **Python:** Uno de estos puertos sigue de cerca el tutorial, mientras que el otro es un reescrito más ambicioso con una línea de comandos interactiva. Ambos utilizan llvmlite como los enlaces a LLVM.
- **Rust and Swift:** Parecía inevitable que tuviéramos portales del tutorial a dos de los idiomas que LLVM ayudó a crear.

Finalmente, los tutoriales también están disponibles en idiomas humanos. Se ha traducido al chino, utilizando el C + original y Python.

Lo que no hace LLVM

Con todo lo que proporciona LLVM, es útil también saber lo que no hace.

Por ejemplo, LLVM no analiza la gramática de un idioma. Muchas herramientas ya hacen ese trabajo, como lex / yacc, flex / bison y ANTLR. El análisis está destinado a ser desacoplado de la compilación de todos modos, por lo que no es sorprendente que LLVM no intente solucionar nada de esto.

LLVM tampoco aborda directamente la cultura más amplia de software en un idioma determinado. Instalar los binarios del compilador, administrar paquetes en una instalación y actualizar la cadena de herramientas, debe hacerlo por su cuenta.

Finalmente, y lo más importante, todavía hay partes comunes de idiomas para las que LLVM no proporciona primitivas. Muchos lenguajes tienen algún tipo de gestión de memoria recolectada en la basura, ya sea como la forma principal de administrar la memoria o como un complemento de estrategias como RAII (que utilizan C + y Rust). LLVM no le proporciona un mecanismo de recolección de basura, pero sí proporciona herramientas para implementar la recolección de basura al permitir que el código se marque con metadatos que facilitan la escritura de recolectores de basura.

Redacción

Más bien lo podemos ver que llvm tiene muchas cosas que nos pueden ayudar a nosotros como alumnos ver cómo funciona el llvm que idiomas maneja o que lenguajes se pueden programar en llvm ahora sí que también su tiempo de compilación de nuestro programa para que podamos ver lo que pedimos realmente ahora si que llvm maneja mas c y c++ son los más utilizables para poder crear nuestro proyecto o un programa y podemos ver que es lo que no hace el llvm en realidad porque podemos ver que no se utiliza tanto el llvm para hacer nuestro programa.