INSTITUTO TECNOLOGICO DE IZTAPALAPA I

INGENIERIA EN SISTEMAS COMPUTACIONALES

Reportes de Apuntes Semanales del

19 AL 23 DE ABRIL 2021

Presenta:

PEREZ ARMAS FAUSTO ISAAC

No. De control:

181080037

ASESOR INTERNO:

M.C. ABIEL TOMAS PARRA HERNANDEZ

CIUDAD DE MEXICO **JUNIO/2021**

INDICE

Actividades semana abril 19-23, 2021

Actividades individuales

Introduction" y "Compiler Structure". Capítulo 1 del libro "Engineering a Compiler" de Cooper y Torczon (2012).

INTRODUCCIÓN

El papel de la computadora en la vida diaria crece cada año. Con el surgimiento de la Internet, las computadoras y el software que se ejecuta en ellas brindan comunicaciones, noticias, entretenimiento y seguridad. Las computadoras integradas han cambiado las formas en que construimos automóviles, aviones, teléfonos, televisores y radios. La computación ha creado categorías de actividad completamente nuevas, desde videojuegos a redes sociales. Las supercomputadoras predicen el clima diario y el curso de violentas tormentas. Las computadoras integradas sincronizan los semáforos y entregue el correo electrónico a su bolsillo. Todas estas aplicaciones informáticas se basan en programas informáticos de software. que construyen herramientas virtuales sobre las abstracciones de bajo nivel proporcionadas por el hardware subyacente. Casi todo ese software es traducido por una herramienta llamado compilador. Un compilador es simplemente un programa de computadora que trans- Compilador un programa de computadora que traduce otros programas de computador lates otros programas informáticos para prepararlos para su ejecución.

Hoja de ruta conceptual

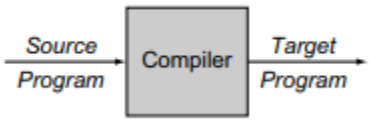
Un compilador es una herramienta que traduce software escrito en un idioma a otro idioma. Para traducir texto de un idioma a otro, la herramienta debe comprender tanto la forma o la sintaxis como el contenido o el significado del idioma de entrada. Necesita comprender las reglas que gobiernan la sintaxis y el significado en el lenguaje de salida. Finalmente, necesita un esquema para mapear contenido. del idioma de origen al idioma de destino. La estructura de un compilador típico se deriva de estas simples observaciones. El compilador tiene una interfaz para manejar el lenguaje fuente. Tiene espalda terminar para tratar con el idioma de destino. Conectando la parte delantera y trasera Al final, tiene una estructura formal para representar el programa en una forma intermedia cuyo significado es en gran medida independiente de cualquiera de los dos idiomas. A mejorar la traducción, un compilador a menudo incluye un optimizador que analiza y reescribe esa forma intermedia.

Descripción general

Los programas de computadora son simplemente secuencias de operaciones abstractas escritas en un lenguaje de programación: un lenguaje formal diseñado para expresar computación. Los lenguajes de programación tienen propieda des y significados rígidos, como opuesto a los lenguajes naturales, como el chino o el portugués. Programación los lenguajes están diseñados para la expresividad, la concisión y la claridad. Natural los idiomas permiten la ambigüedad. Los lenguajes de programación están diseñad os para evitar ambigüedad; un programa ambiguo no tiene sentido. Lenguajes de programación están diseñados para especificar cálculos, para registrar la secuencia de acciones que realizar alguna tarea o producir algunos resultados.

Los lenguajes de programación están, en general, diseñados para permitir que los humanos expresen cálculos como secuencias de operaciones. Procesadores de computadora, en adelante De nominados procesadores, microprocesadores o máquinas, están diseñados para ejecutar secuencias de operaciones. Las operaciones que implementa un procesador tienen, en su mayor parte, un nivel de abstracción mucho más bajo que los especificados en un lenguaje de programación. Por ejemplo, un lenguaje de programación normalmente incluye una forma concisa de imprimir un número en un archivo. Ese soltero La declaración del lenguaje de programación debe traducirse literalmente a cientos de las operaciones de la máquina antes de que pueda ejecutarse.

La herramienta que realiza tales traducciones se llama compilador. El compilador toma como entrada un programa escrito en algún lenguaje y produce como salida un programa equivalente. En la noción clásica de un compilador, la salida El programa se expresa en las operaciones disponibles en algún procesador específico, a menudo llamado la máquina de destino. Visto como una caja negra, un compilador podría se parece a esto:



Los lenguajes "fuente" típicos pueden ser c, c ++, fortran, Java. El idioma "de destino" suele ser el conjunto de instrucciones de algún procesador. Conjunto de instrucciones El conjunto de operaciones soportadas por un procesador; el diseño general de un conjunto de instrucciones es a menudo llamada arquitectura de conjunto de instrucciones o ISA. Algunos compiladores producen un programa de destino escrito en un lenguaje de programación orientado a humanos en lugar del lenguaje ensamblador de alguna computadora. Los programas que producen estos compiladores requieren una traducción adicional antes
pueden ejecutarse directamente en una computadora. Muchos compiladores de investigación producen

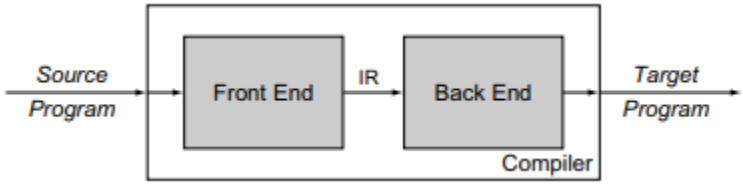
Programas en C como salida. Debido a que los compiladores de C están disponibles en la mayoría de computadoras, esto hace que el programa de destino sea ejecutable en todos esos sistemas, a costa de una compilación adicional para el objetivo final. Los compiladores que s e dirigen a lenguajes de programación en lugar del conjunto de instrucciones de una computadora son a menudo llamados traductores fuente a fuente.

Muchos otros sistemas califican como compiladores. Por ejemplo, un programa de composición tipográfica que produce PostScript puede considerarse un compilador. Toma como Ingrese una especificación de cómo debe verse el documento en la página impresa y produce como salida un archivo PostScript. PostScript es simplemente un lenguaje para describir imágenes. Porque el programa de tipografía toma un ejecutable, especificación y produce otra especificación ejecutable, es un compilador.

El código que convierte PostScript en píxeles suele ser un intérprete, no un compilador. Un intérprete toma como entrada una especificación ejecutable y produce como salida el resultado de ejecutar la especificación.

ESTRUCTURA DEL COMPILADOR

Un compilador es un sistema de software grande y complejo. La comunidad ha sido compiladores de construcción desde 1955, y a lo largo de los años, hemos aprendido muchas lecciones sobre cómo estructurar un compilador. Anteriormente, describimos un compilador como un cuadro simple que traduce un programa fuente en un programa de destino. Realidad, por supuesto, es más complejo que esa simple imagen. Como sugiere el modelo de caja única, un compilador debe comprender el programa fuente que toma como entrada y mapea su funcionalidad al destino máquina. La naturaleza distinta de estas dos tareas sugiere una división del trabajo y conduce a un diseño que descompone la compilación en dos piezas principales: una front end y back end.



La interfaz se centra en comprender el programa en el idioma de origen. La El back-end se enfoca en mapear programas a la máquina de destino. Esta separación de preocupaciones tiene varias implicaciones importantes para el diseño e implementación de compiladores.

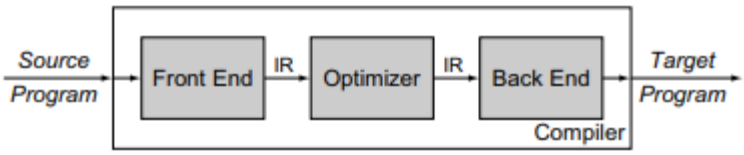
La interfaz debe codificar su conocimiento del programa fuente en algunos Estructura IR para uso posterior por parte del back-end. Esta representación intermedia (ir) Un compilador usa un conjunto de estructuras de datos para representan el código que procesa. Esa forma es llamada representación intermedia, o IR. se convierte en la representación definitiva del compilador del código que está traduciendo. En cada punto de la compilación, el compilador tendrá una representación definitiva. De hecho, puede usar varios IR diferentes a medida que avanza la compilación, pero en cada punto, una representación será la ir definitiva. Pensamos en el ir definitivo como la versión del programa pasada entre independientes Fases del compilador, como el ir pasó del front-end al back-end en el dibujo anterior

En un compilador de dos fases, la interfaz debe garantizar que el programa fuente está bien formado y debe mapear ese código en el archivo ir. El back-end debe mapear el programa ir en el conjunto de instrucciones y los recursos finitos del objetivo máquina. Debido a que el back-end solo procesa lo creado por el front-end, puede sumir que el ir no contiene errores sintácticos o semánticos. El compilador puede realizar varias pasadas sobre la forma ir del código antes emitiendo el programa de destino. Esto debería conducir a un mejor código, ya que el compilador

puede, en efecto, estudiar el código en una fase y registrar detalles relevantes. Luego, en fases posteriores, puede utilizar estos hechos registrados para mejorar la calidad de traducción. Esta estrategia requiere que el conocimiento derivado en el primer paso sea registrado en el ir, donde los pases posteriores pueden encontrarlo y usarlo. Finalmente, la estructura de dos fases puede simplificar el proceso de retargeting la intérprete válida
La tarea de cambiar el compilador para generar el código para un nuevo procesador a menudo se llama Reorientar el compilador. el compilador. Podemos imaginar fácilmente la construcción de múltiples backends para una interfaz única para producir compiladores que acepten el mismo lenguaje, pero se dirijan a diferentes máquinas. Del mismo modo, podemos imaginar las interfaces para diferentes

Descripción general de la compilación

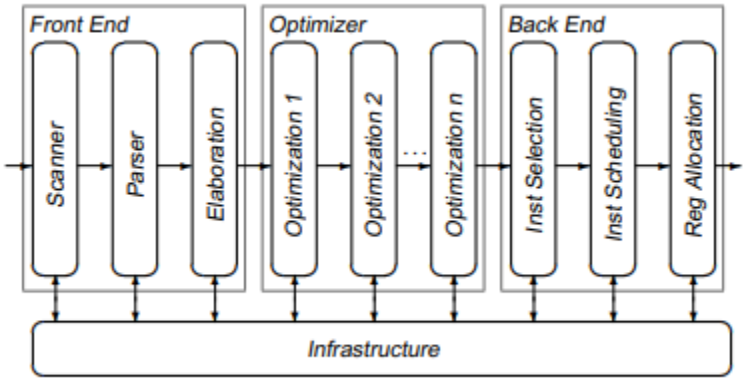
lenguajes que producen el mismo ir y usan un back-end común. Ambas cosas
Los escenarios suponen que un ir puede servir para varias combinaciones de fuentes. y objetivo; en la práctica, tanto los detalles específicos del lenguaje como los específicos de la máquina generalmente encuentran su camino hacia el ir. La introducción de un ir permite agregar más fases a la compilación. Leal escritor del compilador puede insertar una tercera fase entre el front-end y el back Final del optimizador. Esta sección central, u optimizador, toma un programa ir como entrada y La sección central de un compilador, llamada O optimizador, analiza y transforma su IR a mejorarlo, produce un programa ir semánticamente equivalente como salida. Usando el ir como interfaz, el escritor del compilador puede insertar esta tercera fase con un mínimo interrupción en la parte delantera y trasera. Esto conduce al siguiente compilador estructura, denominada compilador trifásico.



El optimizador es un transformador ir a ir que intenta mejorar el programa ir
de alguna manera. (Tenga en cuenta que estos transformadores son, en sí mismos, compiladores de acuerdo con nuestra definición en la Sección 1.1.) El optimizador puede hacer uno o más pasa sobre el ir, analiza el ir y reescribe el ir. El optimizador puede reescribir el ir de una manera que probablemente produzca un programa de destino más rápido desde el back-end o un programa de destino más pequeño desde el back-end. Puede tienen otros objetivos, como un programa que produce menos errores de página o usa menos energía.
Conceptualmente, la estructura trifásica representa la optimización clásica compilador. En la práctica, cada fase se divide internamente en una serie de pases.

La parte delantera consta de dos o tres pasadas que manejan los detalles de reconocer programas válidos en el idioma de origen y producir la información inicial forma del programa. La sección central contiene pases que realizan diferentes optimizaciones. El número y el propósito de estos pases varían de compilador a compilador. El back-end consta de una serie de pases, cada uno de los cuales lo que lleva el programa ir un paso más cerca del conjunto de instrucciones de la máquina de destino. Las tres fases y sus pases individuales comparten una infraestructura. Esta estructura se muestra en la Figura 1.1.

En la práctica, la división conceptual de un compilador en tres fases, una end, una sección intermedia u optimizador, y un back-end, es útil. Los problemas abordados por estas fases son diferentes. La parte delantera se ocupa de comprender el programa fuente y registrar los resultados de su análisis en su forma. La sección del optimizador se centra en mejorar la forma de ir.



■ **FIGURE 1.1** Structure of a Typical Compiler.

Actividades en equipo

1) "Getting Started / Tutorials" de la documentación oficial de LLVM.

El proyecto LLVM tiene múltiples componentes. El núcleo del proyecto en sí mismo se llama "LLVM". Contiene todas las herramientas, bibliotecas y archivos de encabezado necesarios para procesar representaciones intermedias y convertirlas en archivos de objeto. Las herramientas incluyen un ensamblador, un desensamblador, un analizador de código de bits y un optimizador de código de bits. También contiene pruebas de regresión básicas.

Hay muchos proyectos diferentes que componen LLVM. La primera pieza es la suite LLVM. Contiene todas las herramientas, bibliotecas y archivos de encabezado necesarios para usar LLVM. Contiene un ensamblador, desensamblador, analizador de código de bits y optimizador de código de bits. También contiene pruebas de regresión básicas que se pueden utilizar para probar las herramientas LLVM y la interfaz de Clang.

La segunda pieza es la parte delantera de Clang. Este componente compila código C, C ++, Objective C y Objective C ++ en código de bits LLVM. Clang generalmente usa bibliotecas LLVM para optimizar el código de bits y emitir código de máquina. LLVM es totalmente compatible con el formato de archivo de objeto COFF, que es compatible con todas las demás cadenas de herramientas de Windows existentes.

La última parte importante de LLVM, la ejecución de Test Suite, no se ejecuta en Windows, y este documento no lo analiza.

Puede encontrar información adicional sobre la estructura de directorios de LLVM y la cadena de herramientas en la página principal Introducción al sistema LLVM.

Requisitos

Antes de comenzar a utilizar el sistema LLVM, revise los requisitos que se indican a continuación. Esto puede ahorrarle algunos problemas al saber de antemano qué hardware y software necesitará.

Hardware

Cualquier sistema que pueda ejecutar adecuadamente Visual Studio 2017 está bien. El árbol de origen de LLVM y los archivos de objeto, las bibliotecas y los ejecutables consumirán aproximadamente 3 GB.

Software

Necesitará Visual Studio 2017 o superior, con la última actualización instalada.

También necesitará el sistema de compilación CMake, ya que genera los archivos del proyecto que utilizará para compilar.

Si desea ejecutar las pruebas LLVM, necesitará Python. Se sabe que la versión 3.6 y posteriores funcionan. También necesitará herramientas GnuWin32.

No instale el árbol de directorio LLVM en una ruta que contenga espacios (p. Ej.) Ya que el paso de configuración fallará.C:\Documents and Settings\...

Para simplificar el procedimiento de instalación, también puede utilizar Chocolatey como administrador de paquetes. Después de la instalación de Chocolatey, ejecute estos comandos en un shell de administración para instalar las herramientas necesarias:

choco install -y ninja git cmake gnuwin python3

pip3 install psutil

También hay un Dockerfile de Windows con toda la cadena de herramientas de compilación. Esto se puede usar para probar la compilación con una cadena de herramientas diferente a la instalación de su host o para crear servidores de compilación

.