UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Sintaxis y Semántica de los Lenguajes
Comparación GCC vs Clang

VS

Docente: Esp. Ing. Jose Maria Sola
Alumno: Roberto Cuevas

Julio 2025

Indice

Introducción

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

- Rol fundamental del compilador al convertir código fuente en ejecutables eficientes.
- GCC (GNU Compiler Collection) y Clang entre los mas utilizados a nivel global.
- Funciones similares y diferencias significativas en diseño, arquitectura interna, rendimiento, portabilidad y filosofía de desarrollo.
- Esta monografía tiene como objetivo analizar y comparar las características estructurales de GCC y Clang, resaltando diferencias arquitectónicas, en el proceso de compilación y comportamiento práctico frente a casos de C.
- Se incluyen ejemplos muestra como cada compilador maneja determinadas construcciones del lenguaje, facilitando una evaluación crítica de sus ventajas y limitaciones.

Que es GCC

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

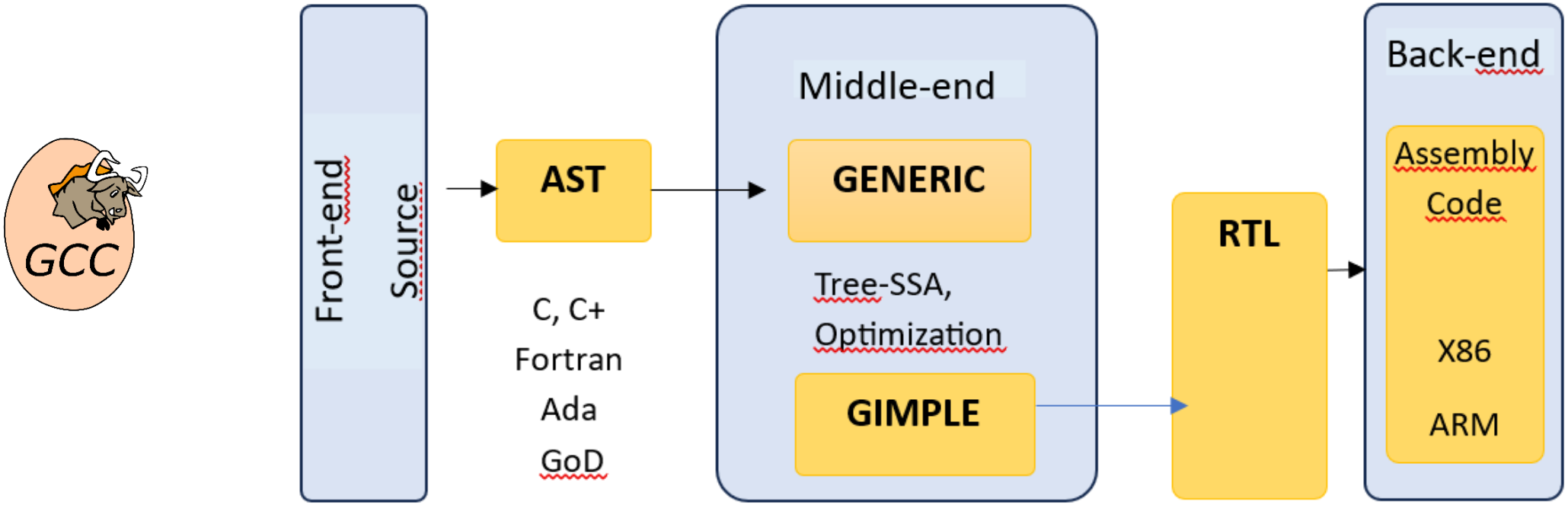
- GCC (GNU Compiler Collection) colección de compiladores desarrollada por el proyecto GNU para C en 1987, mantenida por la Free Software Foundation y comunidad global de desarrolladores.
- Multilenguaje, Soporta C, C++, Fortran, Ada, Go y D, entre otros
- Multiplataforma: Oficicial del SO GNU, actualmente usado en Linux, BSD y SO del tipo Unix.
- Open Source: Bajo licencia GLP, lo que fomenta mejora y distribución libre.
- Maduro y robusto: En desarrollo desde 1987, utilizado en Linux, GNOME y KDE
- Misión: “Desarrollar y mantener una infraestructura de compilación libre, de alta calidad y compatible con estándares.”
- Actualmente alcanzo estabilidad, funcionalidad, aceptación y confiablidad.

Que es CLANG

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

- CLANG es un compilador de C, C++ y Objective-C diseñado como parte LLVM.
- Nació en 2007 como una alternativa moderna a GCC.
- Fuerte popularidad en entornos macOC, iOS y Android.
- Uno de los compiladores mas utilizados, con respaldo de Apple, Google, Meta y AMD.
- Basado en LLVM backend permite análisis optimización y generación en multietapas
- Diagnósticos detallados en mensajes de errores y advertencias mas claros que GCC
- Compilaciones rápidas con poco consumo de memoria, eficiencia
- Arquitectura modular y mantenibilidad, separa frontend del backend
- Interfaz clara para herramientas de análisis
- Mejor integración con herramientas facilita uso en IDEs *(Integrated Development Environment)* y analizadores estáticos.

ARQUITECTURA GCC



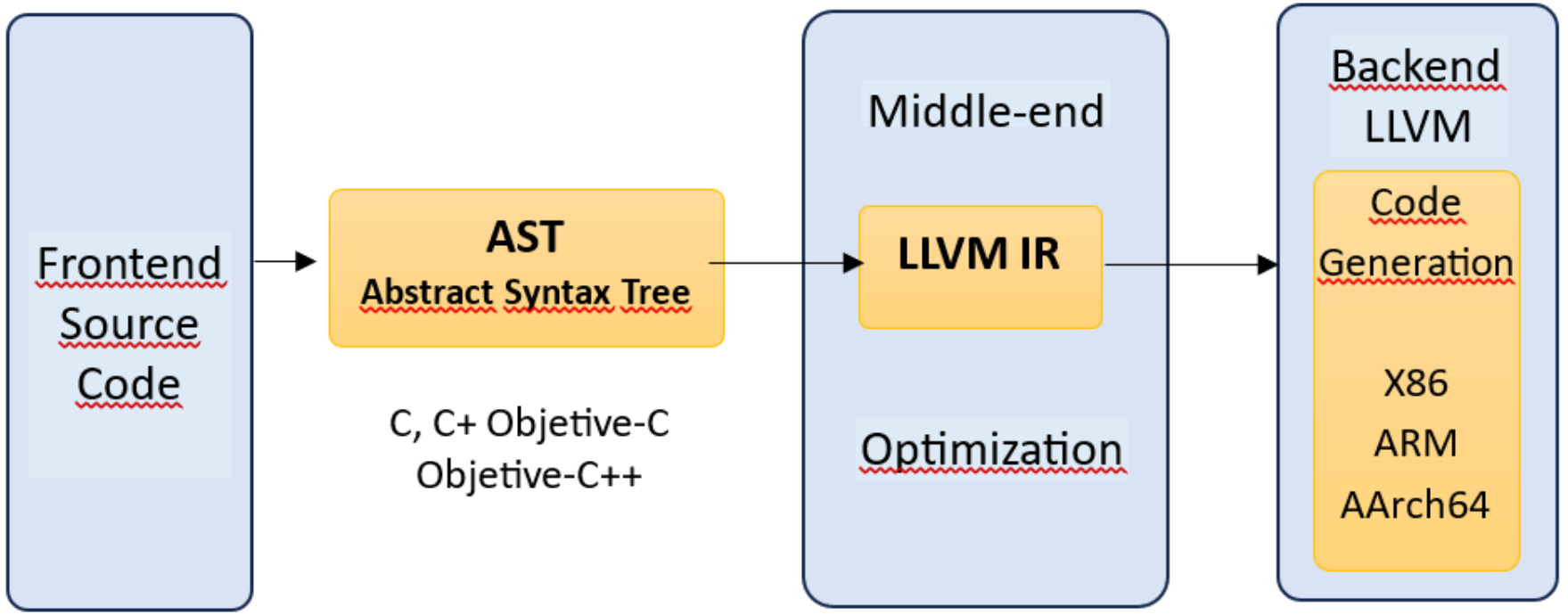
ARQUITECTURA GCC

Frontend: Toma el código fuente (C, C++, Fortran, Ada, etc.) lo convierte a una estructura intermedia, llamado AST (Abstract sintáctic tree), que es la estructura gramatical del código. Forma jerárquica para entender la lógica del programa sin tener en cuenta detalles sintácticos. Luego convierte el AST en la representación GENERIC, común a todos los lenguajes

Middle-end: Transforma GENERIC a GIMPLE, simplifica expresiones y facilita optimizaciones y usa Tree-SSA para optimizaciones avanzadas.

Back-end: Convierte GIMPLE a RTL (Register Transfer Language), una RI de bajo nivel, y produce código ensamblador específico usando usando macros de descripción de máquina

ARQUITECTURA CLANG



ARQUITECTURA CLANG

Frontend: Toma el código fuente (C, C++, Objetive-C, Objetive C++) y lo convierte a una estructura intermedia, llamado AST (Abstract sintáctic tree), que es la estructura gramatical del código. Forma jerárquica para entender la lógica del programa sin considerar sintáxis. Luego convierte el AST en la representación GENERIC, común a todos los lenguajes

Middle-end LLVM IR: Luego del análisis del AST, traduce el código fuente al LLVM IR, un lenguaje de tres niveles (High, Mid y Low Level IR). Este IR permite aplicar un conjunto común de optimizaciones sobre el código..

LLVM Back-end: El LLVM IR se traduce en código máquina mediante el backend de LLVM. Incluye instrucciones específicas de la arquitectura destino (x86, ARM, RISC-V, etc.) Asignación de registros, Generación de código ensamblador, enlazado con bibliotecas y generación de ejecutables.

Diferencias arquitectura y diseño

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Arquitectura y diseño	GCC	Clang / LLVM
Modularidad	Arquitectura integrada: componentes acoplados.	Arquitectura modular: componentes reutilizables.
Representación intermedia (IR)	Usa IR internas (GENERIC, GIMPLE, RTL)	Usa LLVM IR: estándar documentada y compartida
Mensajes de error	Diagnóstico tradicional: mensajes técnicos y menos descriptivos.	Diagnóstico avanzado: mensajes claros, precisos y con sugerencias.
Facilidad de integración	Difícil de integrar debido a diseño monolítico.	Fácil de integrar en IDEs y herramientas - API pública.
Extensibilidad y mantenimiento	Requiere mas recursospara modificar o extender; código más complejo.	Diseñado para extensible: fácil de mantener y expandir a nuevos lenguajes.
Interoperabilidad	Limitada: difícil construir herramientas adicionales externas.	Alta: permite desarrollo de analizadores, refactorizadores, compiladores cruzados.
Separación Frontend/Backend	Frontend y backend unidos; arquitectura menos flexible.	Frontend (Clang) separado del backend (LLVM); adaptable y reutilizable.
Uso como librería (API)	No pensado para ser usado como API externa.	Disponible como librería: se puede usar desde otras aplicaciones fácilmente.

Comparación funcional

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Función del compilador	GCC	Clang
Análisis léxico y sintáctico	Robusto y maduro, con buena detección de errores básicos.	Con mensajes de error más claros y detallados desde el diseño del frontend.
Análisis semántico	Puede requerir flags adicionales (-Wall, -Wextra) para mostrar advertencias específicas.	Tiende a emitir advertencias más explícitas sobre errores lógicos o uso dudoso del código.
Representación intermedia (IR)	Usa múltiples IRs: GENERIC → GIMPLE → RTL.	Usa una única IR: LLVM IR, modular, reutilizable y más uniforme.
Optimización	Aplicada sobre GIMPLE y RTL, con capacidades avanzadas y específicas por arquitectura.	Aplicada sobre LLVM IR, más consistente entre plataformas.
Generación de código objeto	Produce archivos objeto estándar, bien integrados en entornos GNU.	Produce binarios equivalentes, a menudo más compactos gracias a decisiones de diseño en LLVM.
Enlazado (linking)	Usa ld u otras herramientas GNU.	Usa enlazadores del sistema o lld del ecosistema LLVM.
Diagnóstico de errores	Efectivo pero más críptico para usuarios menos experimentados.	Destacado por la claridad, precisión y localización de los mensajes de advertencia y error.
Soporte para análisis estático	Cuenta con herramientas asociadas externas, como gcc-analyzer.	Compatible con clang-tidy y otras herramientas integradas en el ecosistema LLVM.

Comparativa practica

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

- Se utilizó para la comparativa Compiler explorer con gcc 15.1 y clang 20.1.0, excepto en el ejemplo 3 que se utilizó compilación con MSYS2 MinGW 64-bit
- Ejemplo 1 – Comprobación de uso de variables no inicializadas

  Enfoque de Clang en diagnósticos explícitos y el de GCC en robustez de ejecución
- Ejemplo 2 – Comportamiento ante extensiones de GNU (GNU Not Unix)

  Statement expresión ({…}} GCC nativo CLANG compatibilidad con GNU
- Ejemplo 3 – Diferencia en tamaño del ejecutable generado

  GCC 3.25% mayor. Compilacion CLANG significativamente mas rapida
- Ejemplo 4 – Detección de posibles errores lógicos

  Condición lógica contradictoria, Clang mostro advertencia. GCC no detecto
- Ejemplo 5 – Compatibilidad código

  GCC no permite compilar el código. CLANG acepta **auto int** como valido

Conclusión general

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

- **GCC** solida integración con GNU y mayor tolerancia a construcciones de versiones anteriores del lenguaje.
- **Clang** diagnósticos mas precisos, advertencias detalladas, tendencia a generar ejecutables mas eficientes.
- Como conclusión final, importante considerar al elegir un compilador, el rendimiento, en que contexto se va a realizar el proyecto, la portabilidad deseada, el estándar de lenguaje requerido y el control buscado durante el desarrollo.
- Tanto GCC como Clang son herramientas poderosas y activamente mantenidas, y conocer sus diferencias mejora la calidad del código y la eficiencia del desarrollo.

Gracias

Sintaxis y Semántica de los Lenguajes
Comparación GCC vs Clang

VS

Docente: Esp. Ing. Jose Maria Sola