UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Diseño e implementación de un lenguaje de programación y su compilador
Documentación Técnica

Presentado por Adrián Zamora Sánchez
en Universidad de Burgos — 11 de diciembre de 2025
Tutor: César Ignacio García Osorio

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

En este apartado se tiene como objetivo definir y estructurar el plan de proyecto para el desarrollo de un lenguaje de programación y el compilador del mismo, de forma que queden satisfechas sus necesidades y requerimientos técnicos, garantizando la calidad, eficiencia y cumplimiento de unos plazos establecidos. Este plan proporciona un marco de referencia para la gestión del proyecto, incluyendo la planificación temporal así como un estudio de la viabilidad tanto económica como legal.

A.2. Planificación temporal

En este apartado se explica de que forma se aplicará la metodología ágil por excelencia, Scrum, para el control temporal de este proyecto, cabe destacar que al ser un trabajo unipersonal con fin académico puede no reflejar a la perfección el trabajo habitual de esta metodología.

A continuación se muestra una planificación organizada por sprints, donde se prioriza la mejora de un producto mínimo viable (MVP por sus siglas en inglés), de forma que en los primeros sprints se obtiene este MVP y en los próximos sprints se trabaja en ampliar las funcionalidades incrementalmente con cada sprint. Cada dos sprints se realiza una reunión con el tutor, para evaluar el trabajo realizado y planificar los siguientes sprints, teniendo en cuenta lo aprendido anteriormente y las posibles dificultades encontradas.

Así mismo, para mantener un orden durante el desarrollo, se integrará en Github la herramienta ZenHub, la cual ayudará a mantener la planificación

en un tablero estilo Kanban, el cual se integra con las issues y milestones definidas.

A continuación se citan los sprints realizados:

1º Milestone: MVP

Sprint 0 (15/09/2025 - 04/10/2025): Se investiga y sobre herramientas y conceptos relacionados con los lenguajes y compiladores.

Sprint 1 (05/10/2025 - 11/10/2025): Se inicia el repositorio y se trabaja en las primeras dos fases (análisis léxico y sintáctico) del compilador.

Sprint 2 (12/10/2025 - 18/10/2025): Primera reunión con el tutor. Se trabaja en la fase de generación de código intermedio del compilador.

Sprint 3 (19/10/2025 - 24/10/2025): Se añade una funcionalidad para visualización del AST, además, se incluyen mejoras y se madura el trabajo realizado en los sprints anteriores.

2º Milestone: Lenguaje completo

Sprint 4 (25/09/2025 - 01/11/2025): Primera ampliación del MVP, se añaden tipos, declaraciones y asignaciones de variables.

Sprint 5 (02/11/2025 - 08/11/2025): Segunda ampliación del lenguaje, se añaden las funciones y estructura de control if-else.

Sprint 6 (09/11/2025 - 14/11/2025): Tercera ampliación del lenguaje, se añaden bucles while y for. Se añade la fase del compilador responsable de llevar el LLVM IR a código máquina específico de la arquitectura host.

Sprint 7 (15/11/2025 - 21/11/2025): Se añade un runtime mínimo personalizado, se integran funciones de la librería estándar de C como printf, strlen e implementa una librería estándar propia con una función toString. Se añade el tipo que modelará el tiempo.

Sprint 8 (22/11/2025 - 28/11/2025): Se añade la estructura event, que modela la principal característica propia del lenguaje, además se trabaja en refactorizaciones de calidad y seguridad de la fase de análisis semántico del compilador.

Sprint 9 (29/11/2025 - 05/12/2025): Se realizan importantes mejoras de calidad del software. Se añade shadowing de variables y se comienza a hacer pruebas sobre

Sprint 10 (06/12/2025 - 12/12/2025): Se completa una primera versión funcional del runtime, se añaden operadores unitarios ++ y −. Se añade una función print propia del lenguaje.

Sprint 11 (13/12/2025 - 19/12/2025): Se completa el paso de parámetros mediante libffi entre runtime y código compilado. Se mejora la gestión de argumentos y de errores del compilador.

Sprint 12 (20/12/2025 - 26/12/2025):

Sprint 13 (27/12/2025 - 03/01/2025):

Sprint 14 (04/01/2025 - 15/01/2025): El sprint final.

A.3. Estudio de viabilidad

En este apartado analizaremos la viabilidad vista desde el punto de vista de recursos económicos y requisitos para llevarlo a cabo dentro del marco legal. Este análisis puede parecer poco necesario en un proyecto académico, sin embargo, sería vital en un proyecto privado real.

Viabilidad económica

A continuación analizaremos que costes deben ser tenidos en cuenta para el desarrollo de este proyecto, así como una determinación aproximada de los costes específicos y totales.

Coste humano: es el coste del salario que se requiere para el equipo que programará durante el proyecto. Puesto que es un proyecto con un único desarrollador, se estima un sueldo mensual de unos 1500€ brutos.

Coste hardware: este coste representa todo el material que requiere el equipo para funcionar correctamente, este comprende materiales como ordenadores, teclados, ratones, etc. Dando por hecho que en los 5 meses de proyecto solo se alquila el material necesario, podría estimarse un alquiler mensual de 200€ por hardware.

Coste software: estos son costos asociados comúnmente a desarrollos software, sin embargo, con la elección de software libre vamos a evitar este gasto tan común.

Coste total

En este caso de un proyecto de 5 meses, los costes los podemos aproximar de la siguiente forma:

Tipo de coste	Precio aproximado
Coste humano	7500€
Coste hardware	1000€
Coste software	0€
Coste total	**8500€**

Tabla A.1: Costes totales

Viabilidad legal

A continuación se citan todos las herramientas utilizadas y las licencias asociadas a las mismas:

Herramienta	Licencia
Ubuntu	GPL
ANTLR	BDS
LLVM	UIUC
GCC	GPLv3
CMake	BSD
LaTex	LPPL
Doxygen	GPL

Tabla A.2: Licencias

Como podemos observar, todas las licencias de estas herramientas permiten el uso **gratuito** de las mismas, sin restricciones adicionales que afecten a este proyecto.

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este anexo se especifican los requisitos que debe cumplir el proyecto para considerarse completado, así como los casos de uso que se espera que los usuarios hagan de este sistema.

B.2. Objetivos generales

- Comprender el funcionamiento del proceso de compilación de los compiladores modernos.
- Diseñar un lenguaje de programación con un enfoque específico.
- Utilizar la herramienta ANTLR para crear analizadores léxicos y sintácticos.
- Utilizar la herramienta LLVM para pasar un AST a un IR.

B.3. Requisitos funcionales

A continuación se listan los RF (requisitos funcionales) necesarios para completar el proyecto:

RF-1 Capacidad de analizar un texto en el lenguaje fuente: El programa debe ser capaz de aceptar entradas válidas en el lenguaje fuente.

RF-1.1 Gestión de errores léxicos: El programa debe mostrar casos donde se encuentran errores léxicos al analizar la entrada.

RF-1.2 Información de debug léxica: El programa debe ser capaz de ofrecer información sobre los tokens que ha encontrado en el programa mediante un argumento.

RF-2 Capacidad para análisis sintáctico: El programa debe poder analizar sintácticamente el texto recibido siguiendo las reglas de producción definidas.

RF-2.1 Gestión de errores sintácticos: El programa mostrará un error frente a errores sintácticos, mostrando la fuente del error por pantalla.

RF-2.2 Información de debug sintáctica: El programa mostrará información sobre el propio proceso de análisis sintáctico.

RF-3 Creación del AST: El programa generará la AST a partir de la información obtenida en los pasos de análisis léxico y sintáctico.

RF-3.1 Recorrido con patrón visitor: Se implementa un visitor pattern capaz de recorrer de forma ordenada los contextos del parser.

RF-3.2 Recolecta de información de cada estructura: Cada estructura visitada debe almacenar los datos más relevantes de cara a las siguientes fases del compilador.

RF-3.3 Impresión por pantalla del resultado de la AST: Se debe poder mostrar una imagen representativa de los nodos del AST.

RF-4 Análisis semántico: El programa creará la tabla de símbolos y comprobará los tipos en asignaciones y declaraciones.

RF-4.1 Asociación de simbolos y sus datos:

RF-4.2 Gestión de los scopes:

RF-4.3 Comprobaciones de tipos y compatibilidad en operaciones:

RF-5 Generación de código IR: El programa será capaz de crear una representación intermedia en LLVM IR.

RF-5.1 Recorrido con patrón visitor: El programa recorrerá el AST de forma ordenada generando un IR para cada estructura encontrada.

RF-5.2 Optimizaciones: Se deben aplicar las optimizaciones propias de la generación del IR.

RF-6 Optimizaciones:

RF-7 Generación de código máquina:

B.4. Casos de uso

Actores

Los actores principales son los usuarios, que parten de un lenguaje fuente el cual desean ejecutar en un sistema con una determinada arquitectura. Se podría considerar el sistema de archivos como un actor secundario, parte del sistema operativo que interactuaría directamente con los archivos fuente y los ejecutables.

Especificación de los CU

CU-1	**Traducción a código ejecutable**
Versión	1.0
Autor	Adrián Zamora Sánchez
Requisitos asociados	RF-5
Descripción	El programa parte de un código fuente y devuelve un programa ejecutable
Precondición	Existe un fichero con código fuente el cual es correcto léxica y sintácticamente
Acciones	1. Escribir un código fuente 2. Ejecutar el compilador 3. Recibir errores (en este caso hay que volver al paso nº1) o un fichero de salida 4. Se puede ejecutar el fichero de salida
Postcondición	Un fichero ejecutable o un error de compilación y sus detalles
Excepciones	El fichero con el código de entrada no existe, el código fuente es incorrecto
Importancia	Alta

Tabla B.1: CU-1 Traducción a código ejecutable

CU-2	**Visualización de estado**
Versión	1.0
Autor	Adrián Zamora Sánchez
Requisitos asociados	RF-5
Descripción	El programa parte de un código fuente y devuelve un programa ejecutable, adicionalmente se muestra la salida de cada fase del compilador para permitir el estudio de su comportamiento. Para un análisis menos detallado se permite únicamente observar la estructura del programa generado y algunos de sus datos mediante la generación de una imagen del AST
Precondición	Existe un fichero con código fuente el cual es correcto léxica y sintácticamente
Acciones	1. Escribir un código fuente 2. Ejecutar el compilador con la flag –debug o –visualizeAST (segun el nivel de detalle que se requiera) 3. Recibir errores (en este caso hay que volver al paso nº1) o un fichero de salida 4. Se puede ejecutar el fichero de salida
Postcondición	Un fichero ejecutable y datos sobre la compilación o un error de compilación y sus detalles
Excepciones	El fichero con el código de entrada no existe, el código fuente es incorrecto
Importancia	Alta

Tabla B.2: CU-2 Visualización de estado

CU-3	**Traducción a código ejecutable**
Versión	1.0
Autor	Adrián Zamora Sánchez
Requisitos asociados	RF-5
Descripción	El programa parte de un código fuente y devuelve un fichero con IR
Precondición	Existe un fichero con código fuente el cual es correcto léxica y sintácticamente
Acciones	1. Escribir un código fuente 2. Ejecutar el compilador 3. Utilizar el argumento -IR 4. Recibir errores (en este caso hay que volver al paso nº1) o un fichero de salida 5. Se puede abrir y leer el fichero con código IR
Postcondición	Un fichero con IR o un error de compilación y sus detalles
Excepciones	El fichero con el código de entrada no existe, el código fuente es incorrecto
Importancia	Alta

Tabla B.3: CU-3 Traducción a código IR

CU-4	**Comprobación de la validez léxica de un código**
Versión	1.0
Autor	Adrián Zamora Sánchez
Requisitos asociados	RF-5
Descripción	El programa parte de un código fuente y devuelve una cadena con los lexemas detectados
Precondición	Existe un fichero con código fuente
Acciones	1. Escribir un código fuente 2. Ejecutar el compilador 3. Utilizar el argumento -lex 4. Recibir errores (en este caso hay que volver al paso nº1) o una lista de lexemas en la salida principal
Postcondición	Una lista de lexemas o un error durante el análisis léxico y sus detalles
Excepciones	El fichero con el código de entrada no existe
Importancia	Baja

Tabla B.4: CU-4 Comprobación de la validez léxica de un código

CU-5	**Comprobación de la validez sintáctica de un código**
Versión	1.0
Autor	Adrián Zamora Sánchez
Requisitos asociados	RF-5
Descripción	El programa parte de un código fuente y devuelve una visualización del AST
Precondición	Existe un fichero con código fuente
Acciones	1. Escribir un código fuente 2. Ejecutar el compilador 3. Utilizar el argumento -par 4. Recibir errores (en este caso hay que volver al paso nº1) o un AST lista en la salida principal
Postcondición	Un AST o un error durante el análisis léxico o léxico y sus detalles
Excepciones	El fichero con el código de entrada no existe
Importancia	Baja

Tabla B.5: CU-5 Comprobación de la validez sintáctica de un código

CU-6	**Compilar sin optimizar**
Versión	1.0
Autor	Adrián Zamora Sánchez
Requisitos asociados	RF-5
Descripción	El programa parte de un código fuente y realiza el proceso completo a excepción de las optimizaciones finales, el cual sirve para poder comparar con uno optimizado y analizar este proceso
Precondición	Existe un fichero con código fuente y este es correcto léxica y sintácticamente
Acciones	1. Escribir un código fuente 2. Ejecutar el compilador 3. Utilizar el argumento -basic 4. Recibir errores (en este caso hay que volver al paso nº1) o una salida no optimizada
Postcondición	Un programa ejecutable o un fichero con código IR según se especifique
Excepciones	El fichero con el código de entrada no existe o contiene errores
Importancia	Media

Tabla B.6: CU-6 Compilar sin optimizar

Apéndice C

Especificación de diseño

C.1. Introducción

En esta sección se describe cómo se construirá el sistema desde el punto de vista del diseño. El objetivo es definir la estructura general de la aplicación, los modelos de datos que la sustentan y los procedimientos que regirán su funcionamiento. De esta manera se establece un puente entre los requisitos previamente definidos y la posterior implementación, garantizando que el desarrollo siga una organización clara y coherente.

C.2. Diseño de datos

Estructuras de datos utilizadas

AST

Tabla de símbolos

Diagramas UML de los objetos del sistema

Tipos de datos del lenguaje

C.3. Diseño arquitectónico

A continuación se lista los módulos principales:

Analizador léxico o lexer

Analizador sintáctico o parser

Analizador semántico

Generador de código IR

El siguiente diagrama muestra linealmente una ejecución sin errores, donde se puede visualizar el estado de los datos en cada capa:

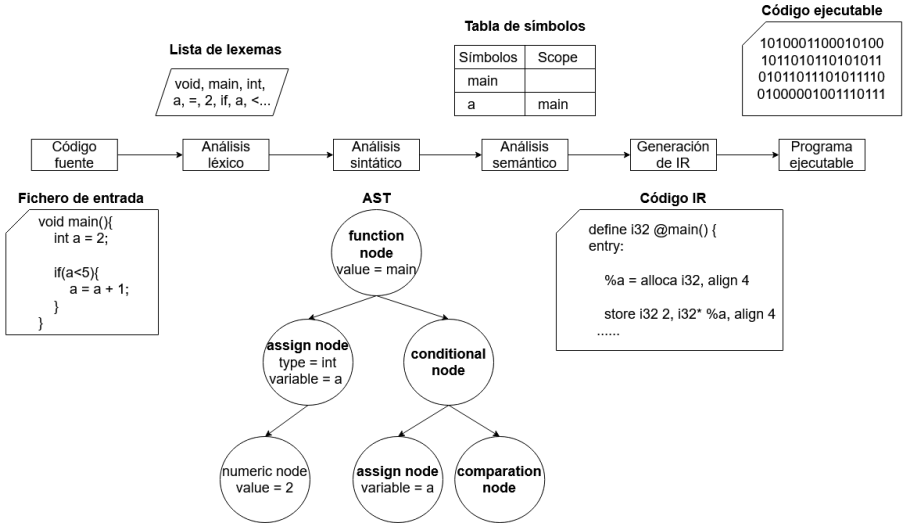

Figura C.1: Diagrama fases del compilador

C.4. Diseño procedimental

A continuación se explican de qué formas realizan sus tareas los elementos del diseño arquitectónico visto anteriormente:

Algoritmo del lexer (NFA → DFA)

Algoritmo parser (Adaptive LL()* / ALL(*)) que primero intenta decisiones con bajo lookahead (SLL), y si falla, usa contexto completo (modo LL)

Algoritmo para generar la AST (visitor pattern sobre contextos ANTLR)

También visitor pattern para recorrer AST en análisis semántico y generación de IR.

Algoritmo de scheduling de eventos

Apéndice D

Documentación técnica de programación

D.1. Introducción

En esta sección se analizará la estructura del trabajo de programación, cómo se estructuran los ficheros de código, cómo se instalan los requisitos, cómo compilar el proyecto, etc.

D.2. Estructura de directorios

A continuación se muestra un diagrama con los principales directorios y ficheros presentes en el proyecto.

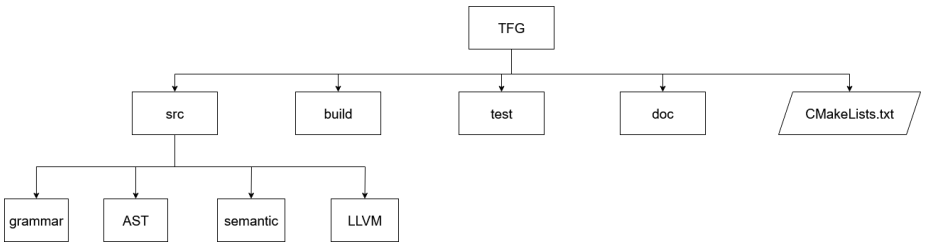

Figura D.1: Diagrama de directorios

D.3. Manual del programador

El objetivo de este manual es ofrecer una visión técnica del código fuente, de manera que cualquier desarrollador que necesite mantener, extender o comprender el sistema pueda hacerlo.

Requisitos previos

Instalación de ANTLR y LLVM

Lenguaje de programación y estilo

El compilador está desarrollado en C++17, empleando un estilo de programación modular y orientado a objetos. Se sigue la convención de nombres en *camelCase* para métodos y variables, y como es estándar de C++, *PascalCase* para clases. El código está documentado con comentarios en formato Doxygen para la generación automática de documentación.

Convenciones internas

- Los ficheros de cabecera se ubican en el directorio `include/`, y las implementaciones en `src/`.
- La definición del lenguaje de ANTLR se encuentra en el directorio `src/grammar/`.
- El AST y las clases de nodos se implementan en `src/ast/`.
- El análisis semántico se implementa en `src/semantic/`.
- El generador de código intermedio se localiza en `src/LLVM/`.
- El sistema de tests se encuentra en el directorio `test/`.
- La documentación del proyecto se encuentra en `doc/`.

D.4. Compilación, instalación y ejecución del proyecto

Compilación

El fichero CMakeLists.txt será el responsable de compilar el proyecto, mediante CMake, herramienta la cual tiene muy buena integración con C++.

Instalación

Ejecución

D.5. Pruebas del sistema

Para un desarrollo de calidad, debemos asegurarnos que cuando introducimos cambios en el sistema, las características anteriores se mantengan funcionales. Para ello se aportan tests unitarios mediante GoogleTest, un framework diseñado por Google para hacer tests unitarios sobre código C++.

Los tests se encuentran en el directorio "testz se encargan de comprobar la correcta aceptación de código fuente, generación de AST, tablas de símbolos y algunos casos de generación de código IR.

Apéndice E

Documentación de usuario

E.1. Introducción

Este anexo tiene el propósito de guiar a un usuario final que desee instalar, configurar y usar el compilador. Se describen los requisitos que debe cumplir el usuario, el proceso de Instalación y las instrucciones fundamentales para poder utilizar el compilador de forma correcta.

E.2. Requisitos de usuarios

Antes de comenzar a utilizar el sistema, el usuario debe contar con los siguientes requisitos:

Conocimientos

- Conocimientos básicos de programación
- Familiaridad con un editor de código
- Conocimiento de uso de programas CLI

Requisitos de hardware

- CPU: 1GHz
- RAM: 1GB
- Espacio en disco: 500MB

Requisitos de software

- Sistema operativo Windows, Linux o macOS.
- Entorno C++

E.3. Instalación

La descarga se realizará desde el repositorio de código en este enlace.

Una completada la descargada se puede colocar el compilador en la ruta que se desee y utilizar esa misma ruta para ejecutar el compilador, o bien introducir la ruta al compilador como variable de entorno en sistemas Windows o como un alias en sistemas Linux, de forma que podamos ejecutar más comodamente el compilador.

Variables de entorno (Windows)

Alias (Linux)

E.4. Manual del usuario

Para compilar, requerimos de tener un fichero con código y podremos ejecutar el compilador con las siguientes opciones:

- **Compilación básica:**
- **Comprobación de validez léxica:**
- **Comprobación de validez sintáctica:**
- **Compilación a IR:**

Apéndice F

Anexo de sostenibilización curricular

F.1. Introducción

En este anexo se abordan los conceptos de sostenibilidad, es decir, la búsqueda de un equilibrio entre el desarrollo económico, la protección del medio ambiente y la justicia social, con el objetivo de garantizar un futuro digno para las próximas generaciones.

En el marco de este Trabajo de Fin de Grado, centrado en el desarrollo de un lenguaje de programación y la construcción de su compilador, resulta complejo reflexionar sobre como herramientas como esta, la cual aparentemente queda fuera de estos debates sociales y ambientales, puede también contribuir a los objetivos de sostenibilidad, sin embargo, hay algunos puntos en los que podemos abordar este tema.

F.2. Principios de sostenibilidad del proyecto

A continuación se tratan los principios de sostenibilidad desde el punto de vista de este proyecto: Principio ético: la programación de un compilador debe orientarse a respetar la transparencia, la seguridad de los datos y la accesibilidad del software. La forma más accesible de presentar este software es con una licencia que lo haga gratuito e impulse a los usuarios a utilizarlo en todo tipo de proyectos.

Principio holístico: el compilador no debe entenderse como un simple traductor de lenguaje humano a lenguaje máquina, sino como una pieza de software que tiene un posible impacto en la eficiencia energética de un sistema informático y que puede afectar a la productividad de los desarrolladores.

Principio de complejidad: un software moderno como este debe ser pensado en dimensiones sociales y ambientales, asegurando que el compilador sea eficiente reduciendo el consumo eléctrico así como evitar que el usuario necesito contar con un hardware de altas prestaciones para su ejecución, lo que a largo plazo puede tener un impacto ambiental.

Principio de glocalización: en el marco de la creación de software, podemos incluir este principio tratando de hacer llegar este software a los usuario en sus lenguas locales y con documentación adaptada a todos los niveles de conocimiento técnico posibles, asegurando sus accesibilidad.

Principio de transversalidad: el criterio de la sostenibilidad no es un factor aislado a algunos de los objetivos del proyecto, sino que debe estar presente durante todo ciclo de diseño así como en el proceso de desarrollo, asegurando todos los principios cubiertos anteriormente.

F.3. Competencias transversales en sostenibilidad

Siguiendo lo indicado en las directrices de la CRUE, se pueden identificar las siguientes competencias transversales vinculadas al desarrollo del compilador:

SOS1: Contextualización crítica del conocimiento. El compilador se ha diseñado no solo como un ejercicio académico, sino también como una herramienta que puede integrarse en contextos reales de desarrollo de software. Esta visión permite relacionar el trabajo con problemáticas más amplias, como la eficiencia energética de los centros de datos o la necesidad de software más accesible.

SOS2: Uso sostenible de recursos. Un compilador eficiente contribuye a que los programas generados consuman menos recursos. Esto tiene un impacto directo en el consumo energético de los dispositivos, lo cual es especialmente relevante en un contexto donde los centros de procesamiento de datos representan una parte significativa del gasto energético global.

SOS3: Participación comunitaria. El software desarrollado puede ponerse a disposición de la comunidad universitaria o profesional, fomentando la

colaboración abierta y el aprendizaje compartido mediante licencias como la MIT, lo que promueve el acceso igualitario al conocimiento.

SOS4: Principios éticos. En la construcción del compilador se ha procurado mantener buenas prácticas de programación, así como transparencia en el funcionamiento de la herramienta, evitando prácticas que puedan considerarse opacas o poco éticas.

F.4. Impacto social y ambiental del proyecto

El impacto de un compilador sobre la sostenibilidad puede analizarse en dos niveles:

Impacto ambiental: la optimización del código y la reducción del consumo de recursos computacionales contribuyen indirectamente a disminuir el gasto energético. Aunque este efecto pueda parecer limitado en un solo proyecto, los programas generador por los compiladores pueden influir directamente en el consumo de energía de millones de dispositivos.

Impacto social: un compilador bien diseñado puede facilitar el acceso a la programación a estudiantes y profesionales de distintas procedencias. La posibilidad de que la herramienta se difunda como recurso abierto fomenta la democratización del conocimiento, reduciendo barreras de entrada en el aprendizaje de lenguajes de programación.

F.5. Conclusión

Aunque a primera vista este proyecto pueda parecer desconectado de las cuestiones de sostenibilidad, un análisis más detallado demuestra que es posible establecer relaciones significativas. La eficiencia computacional, la ética en el diseño de software, la accesibilidad y la potencial contribución al aprendizaje colectivo son aspectos que reflejan cómo la informática puede integrarse en un marco más amplio de desarrollo humano sostenible.

De esta manera, el presente trabajo no solo cumple una función académica y técnica, sino que también se alinea con los compromisos institucionales de la universidad en materia de sostenibilidad, reforzando la idea de que toda actividad de investigación y desarrollo debe contemplar su impacto social, económico y ambiental.

[**?**]

Bibliografía