



INSTITUTO TECNOLÓGICO DE IZTAPALAPA

INGENIERIA EN SISTEMAS COMPUTACIONALES

Nombre del Proyecto:
Flang

Presenta:

Cortez García Jessica Gabriela

161080218 25%

Guerrero Méndez Fabiola

16106147 25%

Martinez Ortiz Areli Susana

161080228 25%

Martinez Rodriguez Gerardo Emmanuel

161080185 25%

Asesor Interno:

Parrá Hernández Abiel Tomás



Índice

Resumen	1
ABSTRACT	2
Introduccion	3
Objetivo General	4
Objetivos Específicos.....	4
Justificación.....	5
Marco Teorico	6
COMPILADOR	6
LLVM.....	6
MLIR	7
FLANG	7
Características Faltantes.	7
Metodologia de Trabajo.	8
Desarrollo e implementación.	9
Resultados.....	14
Se realizo un ejemplo en sublimetet y se corre con cmd+shift+B.....	17
Conclusiones.....	18
Garcia Jessica Gabriela	18
Guerrero Méndez Fabiola Cortez	18
Martinez Ortiz Areli Susana.....	18
Martinez Rodriguez Gererado Emmanuel.....	18
Fuentes de Informacion.	19
https://github.com/flang-compiler/flang	19

Resumen

Flang es un front-end de lenguaje Fortran diseñado para la integración con LLVM y el optimizador LLVM.

Flang + LLVM es una solución Fortran de calidad de producción diseñada para ser instalada conjuntamente y es totalmente interoperable con Clang C ++.

El rendimiento de Flang de un solo núcleo y OpenMP ahora está a la par con GNU Fortran. Flang ha implementado Fortran 2003 y tiene una implementación casi completa de OpenMP a través de la versión 4.5 dirigida a CPU multinúcleo.

A corto plazo, el Flang actual es un compilador de Fortran de calidad de producción compatible con Fortran 2003, algunas características de Fortran 2008 y OpenMP.

Creemos que a partir de 2020, F18 reemplazará a Flang como una implementación completa de Fortran 2018.

Un nuevo front-end para Fortran se anunció en EuroLLVM el 17 de abril de 2018.

El se está escribiendo desde cero utilizando C ++ moderno. Estará estrechamente alineado con las mejores prácticas de LLVM y escrito en el estilo de LLVM y clang.

F18 ha sido aprobado para convertirse en el compilador estándar de Fortran para LLVM y está en proceso de ser adoptado en LLVM propiamente dicho.

Flang actual: el compilador existente de Fortran 2003/2008

El compilador Flang publicado se basa en el compilador comercial Fortran de NVIDIA / PGI. Flang se anunció en 2015. En 2017, el código fuente se publicó en GitHub.

Palabras Clave: Front-end, LLVM, C++, Fortran, OpenMP, F18.



ABSTRACT

Flang is a Fortran language front-end designed for integration with LLVM and the LLVM optimizer.

Flang + LLVM is a production-quality Fortran solution designed to be co-installed and fully interoperable with Clang C ++.

The performance of single-core Flang and OpenMP is now on par with GNU Fortran. Flang has implemented Fortran 2003 and has a near full implementation of OpenMP through version 4.5 targeting multi-core CPUs.

In the short term, the current Flang is a production-grade Fortran compiler that is compatible with Fortran 2003, some Fortran 2008 features, and OpenMP.

We believe that from 2020, F18 will replace Flang as a full implementation of Fortran 2018.

A new front-end for Fortran was announced at EuroLLVM on April 17, 2018.

It is being written from scratch using modern C ++. It will be closely aligned with LLVM best practices and written in the LLVM and clang style.

F18 has been approved to become the Fortran standard compiler for LLVM and is in the process of being adopted into LLVM proper.

Current Flang - The existing Fortran 2003/2008 compiler

The released Flang compiler is based on the commercial NVIDIA / PGI Fortran compiler. Flang was announced in 2015. In 2017, the source code was published on GitHub.

Keywords: Front-end, LLVM, C++, Fortran, OpenMP, F18.

Introducción

Flang es el front-end Fortran de LLVM y es nuevo para la versión LLVM 11.

Flang es todavía un trabajo en progreso para esta versión y se incluye para experimentación y comentarios.

Flang es capaz de analizar un subconjunto completo del lenguaje Fortran y verificar su corrección. Flang aún no puede generar LLVM IR para el código fuente y, por lo tanto, no puede compilar un binario en ejecución.

Flang puede descomprimir el código fuente de entrada en una forma canónica y emitirlo para permitir las pruebas. Flang también puede invocar un compilador de Fortran externo en esta entrada canónica.

El analizador de Flang tiene soporte completo para:

- Fortran 2018
- OpenMP 4.5
- OpenACC 3.0

Objetivo General

Desarrollar una aplicación que nos permita determinar si la estructura de flang corresponde a una sentencia valida en la definición de un lenguaje en particular teniendo en cuenta el contexto sintáctico y semántico . A si mismo estará capacitado para proponer nuevas formas estructurales en la definición de lenguajes de programación.

Objetivos Específicos

- Comprender el funcionamiento de los autómatas para el reconocimiento o aceptación de cadenas o instrucciones que forman parte o son generadas por un lenguaje.
- Diseñar e implementar un analizador semántico.
- Diseñar e implementar un analizador sintactico.

Justificación

El presente proyecto se enfoca en mejorar los compiladores y darles una mayor calidad así como tener una reducción de tiempos aunque estos aun se encuentran en fases de pruebas.

Así el presente proyecto nos muestra que en esta fase de la elaboración no es compatible aun con Windows, también nos abre las puertas a poder cambiar valores y códigos el cual para los jóvenes programadores es un gran incentivo para desafiar sus habilidades en el desarrollo de nuevas tecnologías.

Marco Teorico

COMPILADOR

Es un Software que traduce un programa escrito en un lenguaje de programación de alto nivel (C / C ++, COBOL, etc.) en lenguaje de máquina. Un compilador generalmente genera lenguaje ensamblador primero y luego traduce el lenguaje ensamblador al lenguaje máquina. Una utilidad conocida como «enlazador» combina todos los módulos de lenguaje de máquina necesarios en un programa ejecutable que se puede ejecutar en la computadora.

El primer compilador del lenguaje de alto nivel FORTRAN se desarrolló entre 1954 y 1957 en IBM por un grupo dirigido por John Backus. Un compilador es uno de los pilares de la programación y de cómo entender la comunicación entre un lenguaje de alto nivel y una máquina. Al poder conocer el funcionamiento de este paso intermedio nos permitirá desarrollar y programar de una forma más precisa los lenguajes de alto nivel.

LLVM

Es una infraestructura para desarrollar compiladores, escrita a su vez en el lenguaje de programación C++, que está diseñada para optimizar el tiempo de compilación, el tiempo de enlazado, el tiempo de ejecución y el "tiempo ocioso" en cualquier lenguaje de programación que el usuario quiera definir. Implementado originalmente para compilar C y C++, el diseño agnóstico de LLVM con respecto al lenguaje, y el éxito del proyecto han engendrado una amplia variedad de lenguajes, incluyendo Objective-C, Fortran, Ada, Haskell, bytecode de Java, Python, Ruby, ActionScript, GLSL, Clang, Rust, Gambas y otros.

El nombre "LLVM" era en principio las iniciales de "**Low Level Virtual Machine**", pero esta denominación causó una confusión ampliamente difundida, puesto que las máquinas virtuales son solo una de las muchas cosas que se pueden construir con LLVM. Cuando la extensión del proyecto se amplió incluso más, LLVM se convirtió en un proyecto paraguas que incluye una multiplicidad de otros compiladores y tecnologías de bajo nivel, haciendo el nombre aún menos adecuado. Por tanto, el proyecto abandonó las iniciales. Actualmente, LLVM es una "marca" que se aplica al proyecto paraguas, la representación intermedia LLVM, el depurador LLVM, la biblioteca estándar de C++ definida por LLVM,

MLIR

El proyecto MLIR define una representación intermedia (IR) común que unifica la infraestructura necesaria para ejecutar modelos de aprendizaje automático de alto rendimiento en TensorFlow y en marcos de trabajo de AA similares. Este proyecto incluirá la aplicación de técnicas de HPC, junto con la integración de algoritmos de búsqueda, como el aprendizaje por refuerzo. El objetivo de MLIR es reducir el costo para incorporar hardware nuevo y mejorar la usabilidad para los usuarios de TensorFlow.

Nos beneficiamos de la experiencia obtenida en la construcción de otros IR (LLVM IR, XLA HLO y Swift SIL) al construir MLIR. El marco MLIR fomenta las mejores prácticas existentes, por ejemplo, escribir y mantener una especificación de IR, construir un verificador de IR, brindar la capacidad de volcar y analizar archivos MLIR en texto, escribir pruebas unitarias extensas con la herramienta FileCheck y construir la infraestructura como un conjunto de bibliotecas modulares que se pueden combinar de nuevas formas.

FLANG

De forma predeterminada, Flang analizará el archivo Fortran y hello.f90 luego lo descomprimirá en un archivo fuente canónico de Fortran. Flang luego invocará un compilador externo de Fortran para compilar este archivo fuente y vincularlo, colocando el ejecutable resultante en formato hello.bin.

Para especificar el compilador externo de Fortran, configure la F18_FCvariable de entorno con el nombre del binario del compilador y asegúrese de que esté en su PATH. El valor predeterminado de F18_FCes gfortran.

Cuando se invoca sin entrada de fuente, Flang esperará la entrada en stdin. Cuando se invoca de esta manera, Flang realiza las mismas acciones que si se llama con y no invoca .-fdebug-measure-parse-tree -funparseF18_FC

[Características Faltantes.](#)

Flang no es compatible con plataformas Windows.

Metodología de Trabajo.

LEAN: está configurado para que pequeños equipos de desarrollo muy capacitados elaboren cualquier tarea en poco tiempo. Los activos más importantes son las personas y su compromiso, relegando así a un segundo plano el tiempo y los costes. El aprendizaje, las reacciones rápidas y potenciar el equipo son fundamentales.

Se escogió dicha metodología ya que a diferencia de las metodologías tradicionales contamos con poco tiempo, además; por el tiempo de desarrollo nos convendría ya que es un proyecto escolar como dice la metodología el aprendizaje, las reacciones rápidas y el potenciar el equipo son fundamentales.

Además, que en las metodologías ágiles se ubican 4 etapas que son:

- **Planteamiento**
- **Requerimientos**
- **Iteración(es)**
- **Puesta en marcha**



a diferencia de una metodología tradicional la cual consta de 6 etapas, las cuales son:

- **Planteamiento**
- **Análisis**
- **Diseño**
- **Programación**
- **Pruebas**
- **Puesta en marcha**



por consiguiente, fue que se escogió **Lean** como metodología para el proyecto.

Fuente: <https://blog.becas-santander.com/es/metodologias-desarrollo-software.html>

Desarrollo e implementación.

INSTALACION.

Descargamos VisualStudio Code(dependerá del SO la versión y los Requisitos)

<https://code.visualstudio.com/download>



Una vez descargado tuvimos que instalar HomeBrew para instalar gcc que incluye gfortran

https://brew.sh/index_es

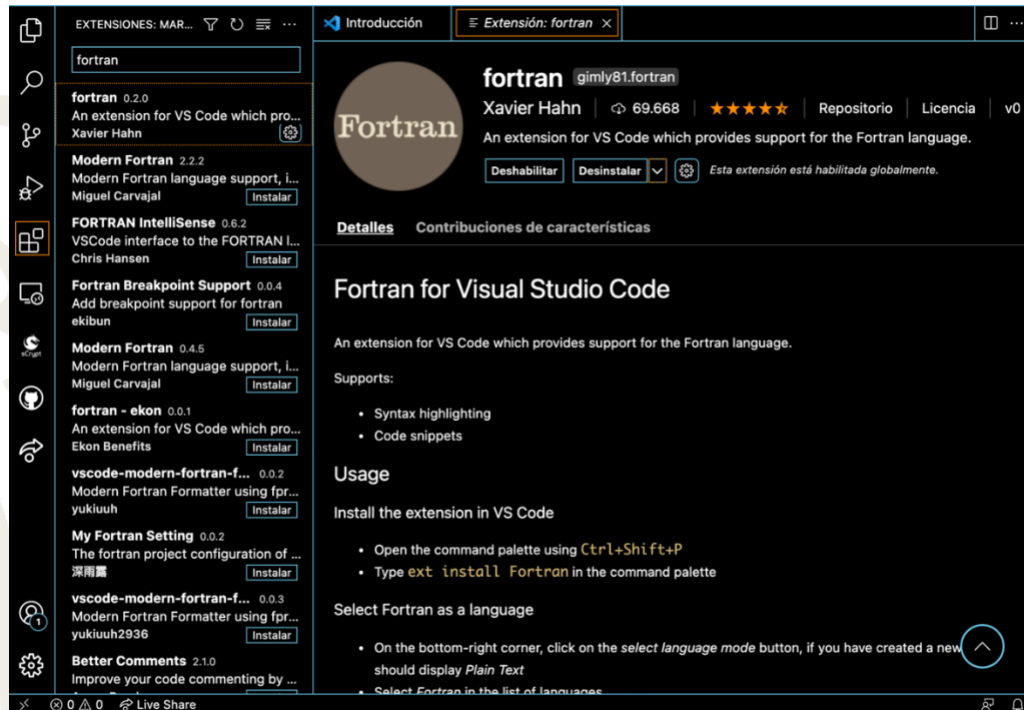
```
/bin/bash -c "$(curl -fsSL
```

```
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Con la línea anterior de comando es como se instala HomeBrew.

```
gera — git — bash -c #!/bin/bash\012set -u\012\012abort() { \012 printf "%...
* [new tag]          3.0.7      -> 3.0.7
* [new tag]          3.0.8      -> 3.0.8
* [new tag]          3.0.9      -> 3.0.9
* [new tag]          3.1.0      -> 3.1.0
* [new tag]          3.1.1      -> 3.1.1
* [new tag]          3.1.10     -> 3.1.10
* [new tag]          3.1.11     -> 3.1.11
* [new tag]          3.1.12     -> 3.1.12
* [new tag]          3.1.2      -> 3.1.2
* [new tag]          3.1.3      -> 3.1.3
* [new tag]          3.1.4      -> 3.1.4
* [new tag]          3.1.5      -> 3.1.5
* [new tag]          3.1.6      -> 3.1.6
* [new tag]          3.1.7      -> 3.1.7
* [new tag]          3.1.8      -> 3.1.8
* [new tag]          3.1.9      -> 3.1.9
* [new tag]          3.2.0      -> 3.2.0
HEAD is now at b273a6332 Merge pull request #11570 from Homebrew/dependabot/bund
ler/Library/Homebrew/rubocop-rails-2.11.0
==> Tapping homebrew/core
remote: Enumerating objects: 977135, done.
remote: Counting objects: 100% (440/440), done.
remote: Compressing objects: 100% (198/198), done.
Receiving objects: 76% (747638/977135), 323.22 MiB | 227.00 KiB/s
```

Mientras se instala HomeBrew decargamos la extension de Fortran en VSCode.



Una vez completado el proceso de Homebrew se instala gcc

```
MacBook-Pro-de-Gera:~ gera$ exit
logout
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.
Deleting expired sessions...16 completed.

[Proceso completado]
```



```
[MacBook-Pro-de-Gera:~ gera$ brew install gcc]
==> Downloading https://ghcr.io/v2/homebrew/core/gmp/manifests/6.2.1
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/gmp/blobs/sha256:6a44705536f25c
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/isl/manifests/0.24
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/isl/blobs/sha256:d8c7026042e122
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/mpfr/manifests/4.1.0
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/mpfr/blobs/sha256:1e8eb0326f62d
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/libmpc/manifests/1.2.1
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/libmpc/blobs/sha256:754667644cc
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/zstd/manifests/1.5.0
#=#=#
```

Instalamos el gdb

```
[MacBook-Pro-de-Gera:~ gera$ brew install gdb]
==> Downloading https://ghcr.io/v2/homebrew/core/gdbm/manifests/1.19
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/gdbm/blobs/sha256:3581501b051db
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/mpdecimal/manifests/2.5.1
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/mpdecimal/blobs/sha256:255b6226
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/openssl/1.1/manifests/1.1.1k
##### 100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/openssl/1.1/blobs/sha256:17d94c
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
##### 35.4%
```

Una vez Instalado el gdb pasamos a visual code a checar si ya está listo

```
==> Installing gdb
==> Pouring gdb--10.2.big_sur.bottle.tar.gz
==> Caveats
gdb requires special privileges to access Mach ports.
You will need to codesign the binary. For instructions, see:

  https://sourceware.org/gdb/wiki/BuildingOnDarwin

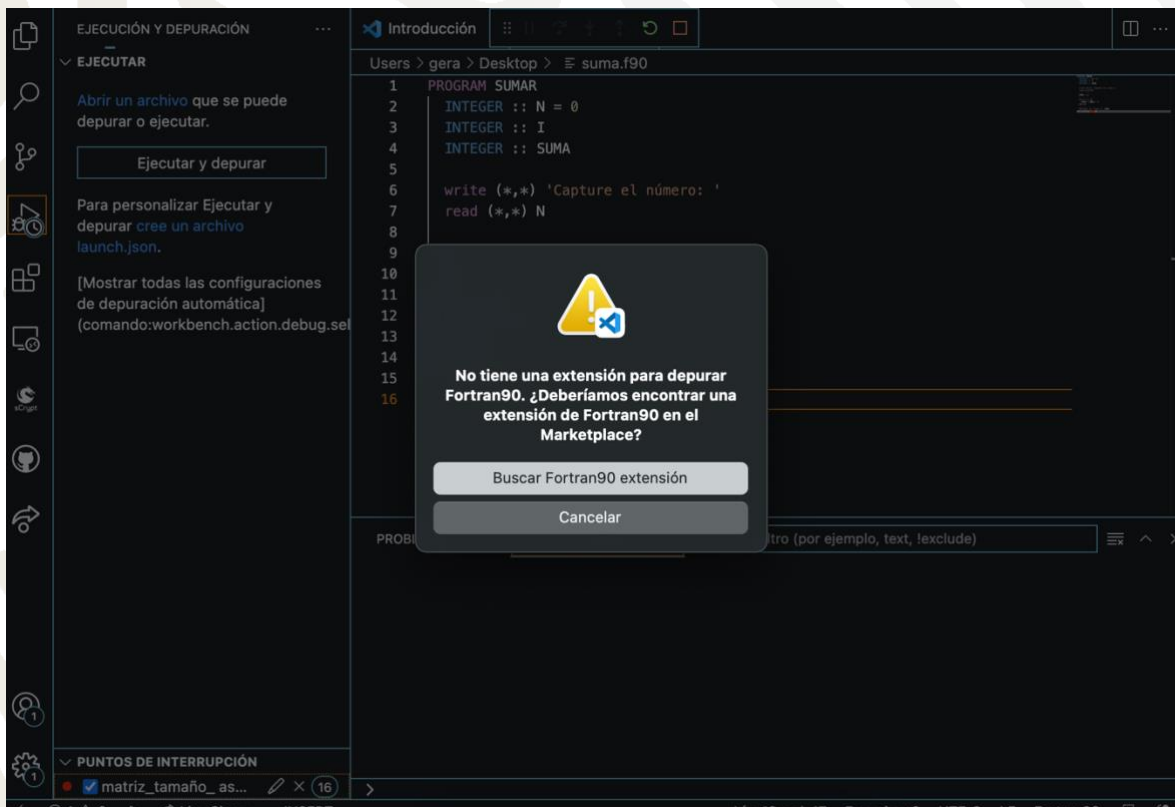
On 10.12 (Sierra) or later with SIP, you need to run this:

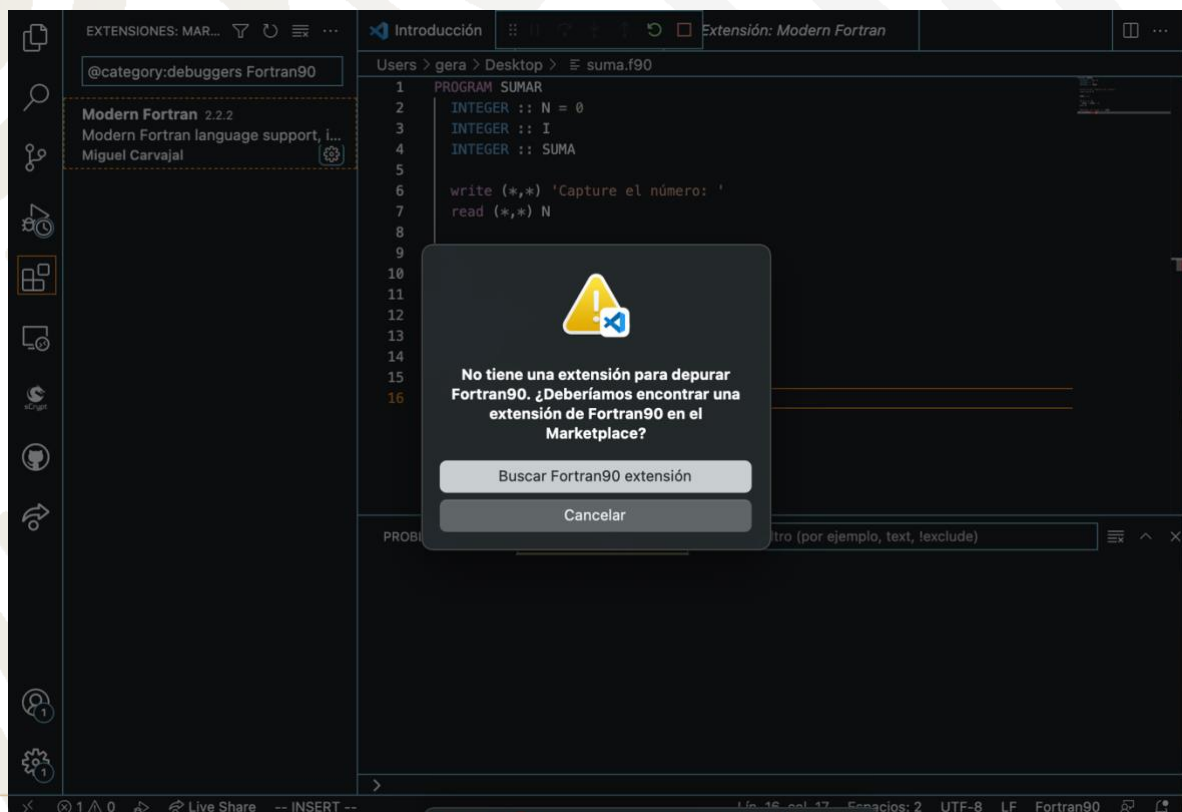
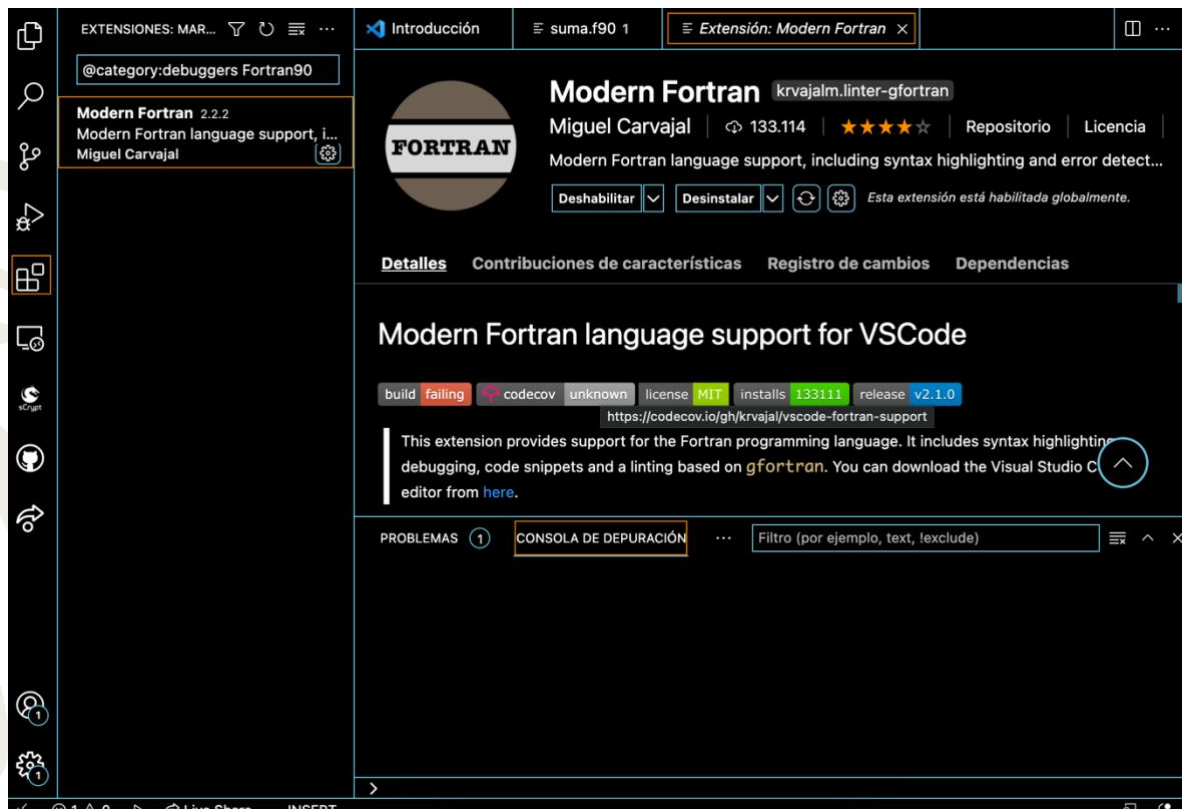
  echo "set startup-with-shell off" >> ~/.gdbinit
==> Summary
📦 /usr/local/Cellar/gdb/10.2: 57 files, 28.3MB
==> Caveats
==> gdb
gdb requires special privileges to access Mach ports.
You will need to codesign the binary. For instructions, see:

  https://sourceware.org/gdb/wiki/BuildingOnDarwin

On 10.12 (Sierra) or later with SIP, you need to run this:

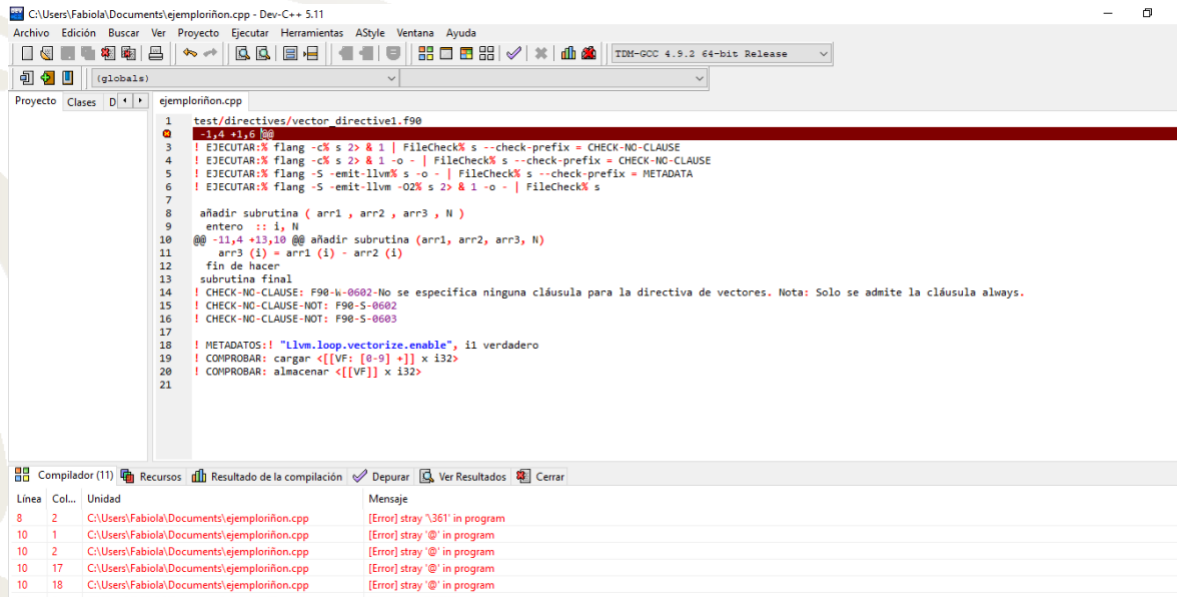
  echo "set startup-with-shell off" >> ~/.gdbinit
MacBook-Pro-de-Gera:~ gera$
```





Resultados.

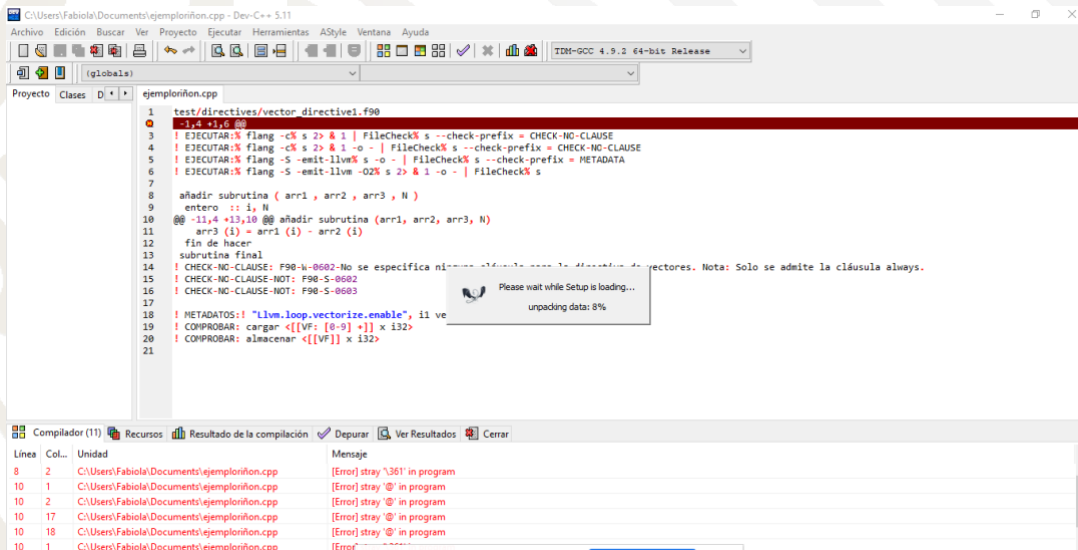
Se realizó la instalación de Dev C++ para realizar un test de un programa de prueba, cabe mencionar que es un programa experimental el cual no asegura el 100% su funcionalidad.



El código fuente en `ejemplorion.cpp` incluye directivas de compilación para LLVM, una función `añadir_subrutina` y una rutina principal que utiliza la función. El panel de mensajes de compilación muestra los siguientes errores:

Línea	Col.	Unidad	Mensaje
8	2	C:\Users\Fabiola\Documents\ejemplorion.cpp	[Error] stray '\361' in program
10	1	C:\Users\Fabiola\Documents\ejemplorion.cpp	[Error] stray '@' in program
10	2	C:\Users\Fabiola\Documents\ejemplorion.cpp	[Error] stray '@' in program
10	17	C:\Users\Fabiola\Documents\ejemplorion.cpp	[Error] stray '@' in program
10	18	C:\Users\Fabiola\Documents\ejemplorion.cpp	[Error] stray '@' in program

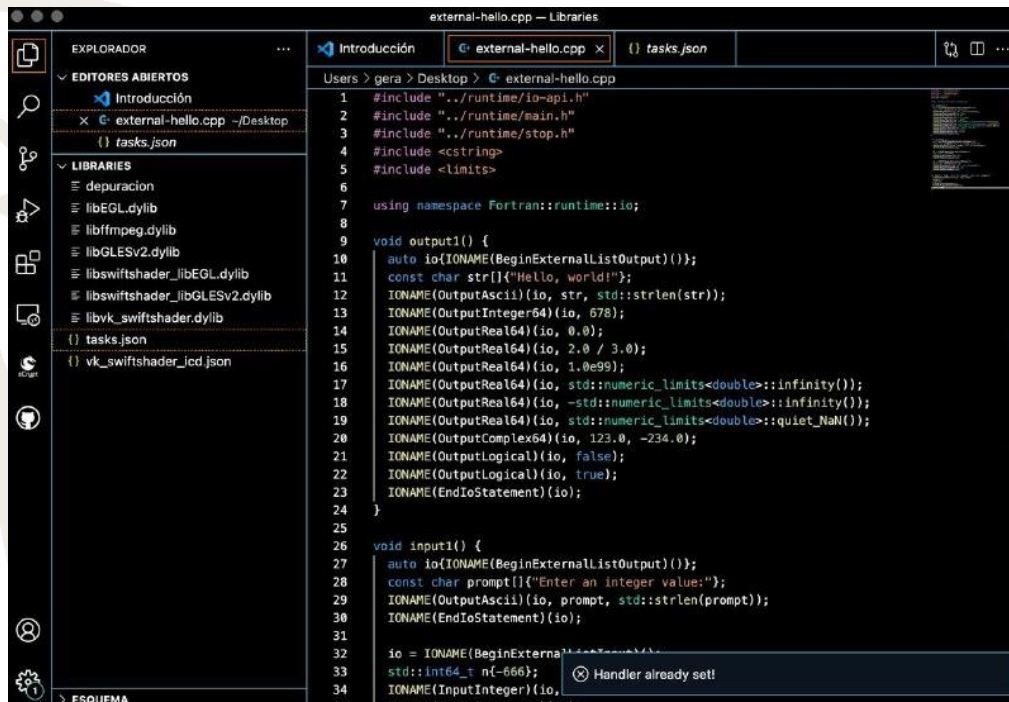
Al realizar las pruebas arrojo error en el programa.



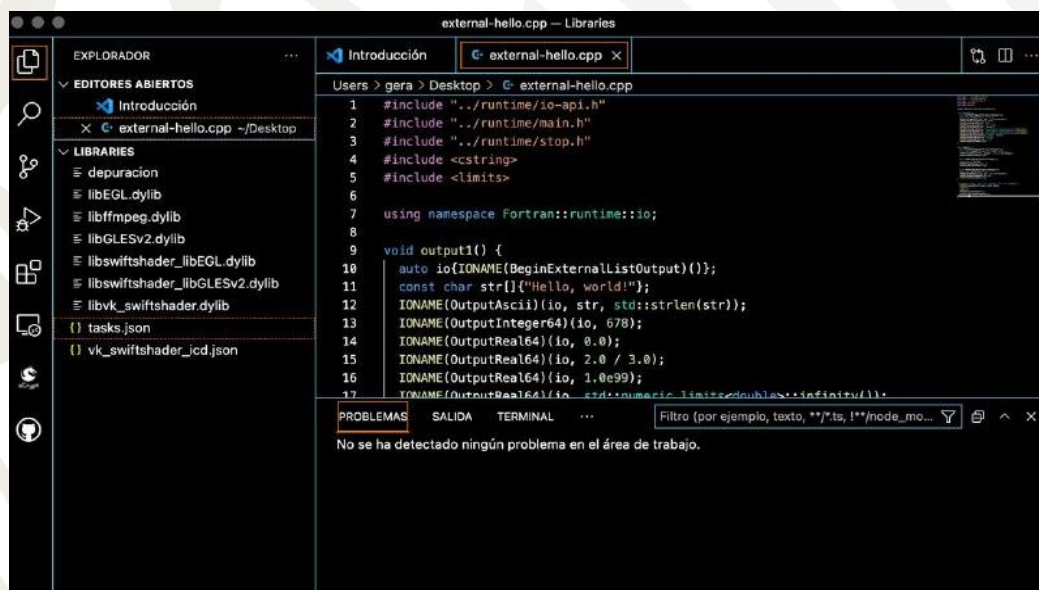
El código fuente es idéntico al anterior. Durante la ejecución, se muestra un mensaje de error: "Please wait while Setup is loading... unpacking data: 8%". El panel de mensajes de compilación muestra los mismos errores que en la imagen anterior.

Se intentó ejecutar el programa en LLVM pero no se logró ejecutar.

Una vez instalado visual Studio Code compilamos un “Hola mundo” para realizar pruebas necesarias y así demostrar el funcionamiento de la interfaz.



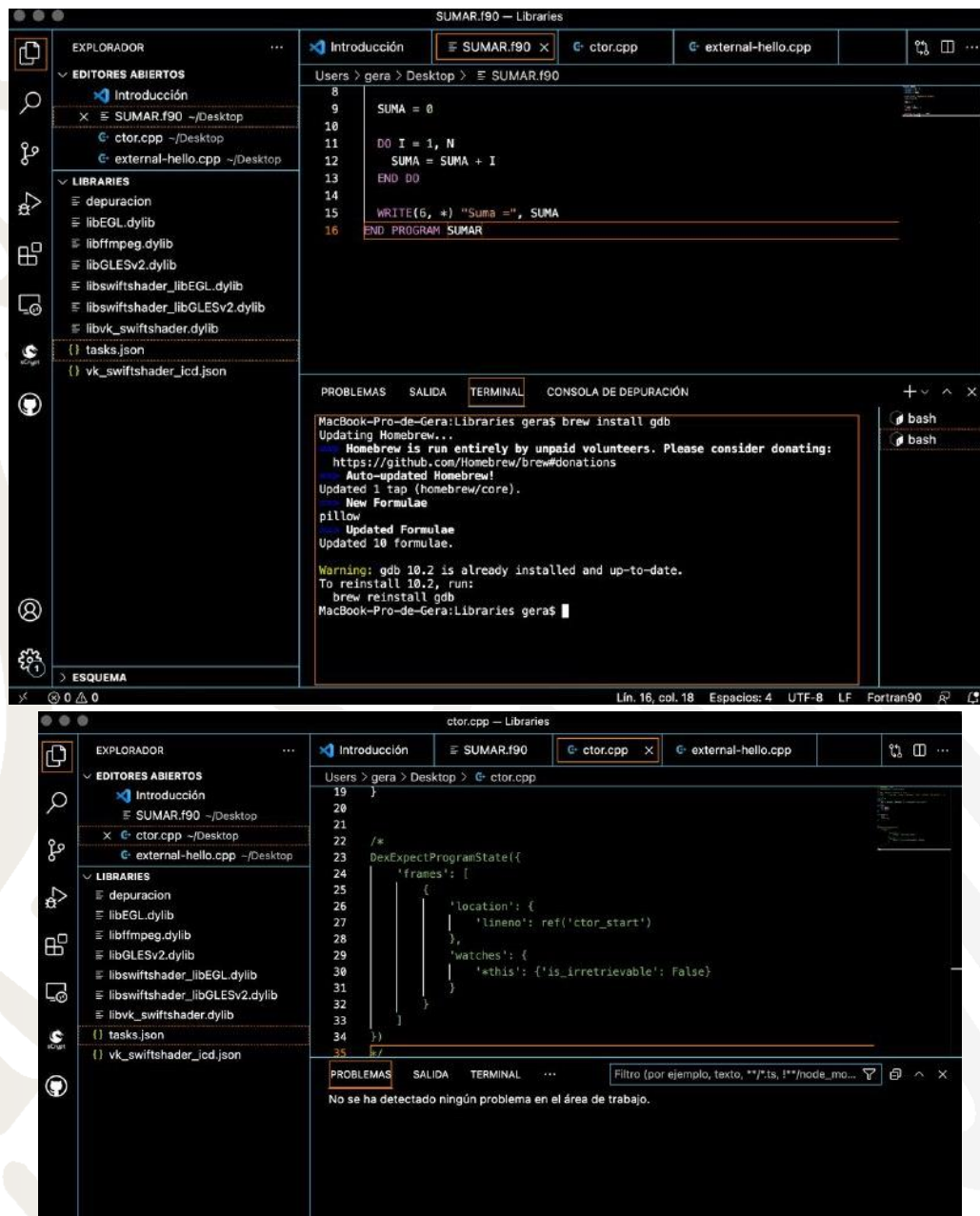
```
1 #include "../runtime/io-api.h"
2 #include "../runtime/main.h"
3 #include "../runtime/stop.h"
4 #include <cstring>
5 #include <limits>
6
7 using namespace Fortran::runtime::io;
8
9 void output1() {
10     auto io(IONAME(BeginExternalListOutput()));
11     const char str[]{"Hello, world!"};
12     IONAME(OutputAscii)(io, str, std::strlen(str));
13     IONAME(OutputInteger64)(io, 678);
14     IONAME(OutputReal64)(io, 0.0);
15     IONAME(OutputReal64)(io, 2.0 / 3.0);
16     IONAME(OutputReal64)(io, 1.0e99);
17     IONAME(OutputReal64)(io, std::numeric_limits<double>::infinity());
18     IONAME(OutputReal64)(io, -std::numeric_limits<double>::infinity());
19     IONAME(OutputReal64)(io, std::numeric_limits<double>::quiet_NaN());
20     IONAME(OutputComplex64)(io, 123.0, -234.0);
21     IONAME(OutputLogical)(io, false);
22     IONAME(OutputLogical)(io, true);
23     IONAME(EndIoStatement)(io);
24 }
25
26 void input1() {
27     auto io(IONAME(BeginExternalListOutput()));
28     const char prompt[]{"Enter an integer value:"};
29     IONAME(OutputAscii)(io, prompt, std::strlen(prompt));
30     IONAME(EndIoStatement)(io);
31 }
32
33 io = IONAME(BeginExternalListOutput());
34 std::int64_t n(-666);
35 IONAME(InputInteger)(io,
```



```
1 #include "../runtime/io-api.h"
2 #include "../runtime/main.h"
3 #include "../runtime/stop.h"
4 #include <cstring>
5 #include <limits>
6
7 using namespace Fortran::runtime::io;
8
9 void output1() {
10     auto io(IONAME(BeginExternalListOutput()));
11     const char str[]{"Hello, world!"};
12     IONAME(OutputAscii)(io, str, std::strlen(str));
13     IONAME(OutputInteger64)(io, 678);
14     IONAME(OutputReal64)(io, 0.0);
15     IONAME(OutputReal64)(io, 2.0 / 3.0);
16     IONAME(OutputReal64)(io, 1.0e99);
17     IONAME(OutputReal64)(io, std::numeric_limits<double>::infinity());
18     IONAME(OutputReal64)(io, -std::numeric_limits<double>::infinity());
19     IONAME(OutputReal64)(io, std::numeric_limits<double>::quiet_NaN());
20     IONAME(OutputComplex64)(io, 123.0, -234.0);
21     IONAME(OutputLogical)(io, false);
22     IONAME(OutputLogical)(io, true);
23     IONAME(EndIoStatement)(io);
24 }
25
26 void input1() {
27     auto io(IONAME(BeginExternalListOutput()));
28     const char prompt[]{"Enter an integer value:"};
29     IONAME(OutputAscii)(io, prompt, std::strlen(prompt));
30     IONAME(EndIoStatement)(io);
31 }
32
33 io = IONAME(BeginExternalListOutput());
34 std::int64_t n(-666);
35 IONAME(InputInteger)(io,
```

Al momento de ejecutar el programa no marca ningún error, pero a la vez no nos imprime nada en la pantalla para mostrar el “Hola mundo”.

Se realizo otro ejemplo con la “suma” en fortran



Las imágenes muestran la interfaz de Visual Studio Code configurada para trabajar con Fortran. La primera imagen muestra el archivo `SUMAR.f90` con el siguiente código:

```
8
9
10 SUMA = 0
11 DO I = 1, N
12   SUMA = SUMA + I
13 END DO
14
15 WRITE(6, *) "Suma =", SUMA
16 END PROGRAM SUMAR
```

La terminal muestra el comando `brew install gdb` ejecutado en un Mac, indicando que se ha instalado correctamente.

La segunda imagen muestra el archivo `ctor.cpp` con el siguiente código:

```
19 }
20
21
22 /*
23 DexExpectProgramState{
24   'frames': [
25     {
26       'location': {
27         'lineno': ref('ctor_start')
28       },
29       'watches': {
30         'this': {'is_irretrievable': False}
31       }
32     }
33   ]
34 }
35 */
```

La terminal muestra el mensaje: "No se ha detectado ningún problema en el área de trabajo."

Se realizó un ejemplo en sublimetext y se corre con cmd+shift+B

```
hello.c
1 // REQUIRES: system-windows
2 //
3 // RUN: %dexter --fail-lt 1.0 -w --builder 'clang-cl_vs2015' \
4 // RUN: --debugger 'dbgeng' --cflags '/Z7 /Zi' --ldflags '/Z7 /Zi' -- %s
5
6 #include <stdio.h>
7 int main() {
8     printf("hello world\n");
9     int x = 42;
10    debugbreak(); // DexLabel('stop') x error: implicit declaration of function '__debugbreak'
11 }
12
13 // DexExpectWatchValue('x', 42, on_line=ref('stop'))

/Users/gera/Desktop/hello.c:10:3: error: implicit declaration of function '__debugbreak' is invalid in
C99 [-Werror,-Wimplicit-function-declaration]
    debugbreak(); // DexLabel('stop')
    ^
1 error generated.
[Finished in 178ms with exit code 1]
[shell_cmd: gcc "/Users/gera/Desktop/hello.c" -o "/Users/gera/Desktop/hello" && "/Users/gera/Desktop/hello"]
[dir: /Users/gera/Desktop]
[path: /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Applications/VMware Fusion.app/Contents/Public:/usr/local/share/dotnet:/usr/share/dotnet:/Library/Apple/usr/bin:/Library/Frameworks/Mono.framework/Versions/Current/Commands]
```

```
hello.c
1 // REQUIRES: system-windows
2 //
3 // RUN: %dexter --fail-lt 1.0 -w --builder 'clang-cl_vs2015' \
4 // RUN: --debugger 'dbgeng' --cflags '/Z7 /Zi' --ldflags '/Z7 /Zi' -- %s
5
6 #include <stdio.h>
7 int main() {
8     printf("hello world\n");
9     int x = 42;
10    // DexLabel('stop')
11 }
12
13 // DexExpectWatchValue('x', 42, on_line=ref('stop'))

hello world
[Finished in 551ms]
```

Conclusiones.

Garcia Jessica Gabriela

Es un tanto difícil dar una conclusión concreta a este proyecto ya que si tiene muchas fallas aun y no lográbamos encontrar las herramientas correctas para correr los programas y aunque al final se logro correr algunos programas obtuvimos dificultades ya que en ocasiones simplemente no funcionaban, no obstante es importante decir que si es una gran propuesta que podría lograr muchas cosas.

Guerrero Méndez Fabiola Cortez

Con la presentación de este proyecto se ha dejado en claro varias cosas de las cuales en el transcurso del desarrollo del proyecto, nos dejan especificado cada uno de los componentes y fases de un compilador y también ha permitido elevar la vista y mirar más allá de lo que se ve en el momento a realizar la búsqueda inalcanzable por obtener un resultados. Podemos darnos cuenta que es posible desarrollar un compilador de un lenguaje de programación totalmente adaptado a nuestras necesidades basándonos en los conocimientos previos por desgracia no se pudo ver claramente el objetivo ,pero se aprendió mucho sobre el tema .

Martinez Ortiz Areli Susana

Como conclusión personal trabajar en este proyecto no fue un tema fácil ya que nunca había trabajado en ello, ni me había informado mucho de él, me costó un poco entenderlo y al querer hacer los programas para hacer pruebas nos pedía muchas extensiones para poder ejecutarlo y siendo así tenia que buscar la manera de poderlos conectar con mi sistema operativo.

Martinez Rodriguez Gererado Emmanuel.

En este punto es difícil dar una conclusion, para mi es un lenjuage que a pesar de que dice en el github que es multiplataforma, se requieren diferentes extensiones, al menos en mi caso me fue difícil el correr algun codigo, desde la isntalacion es un tanto difícil, ya que no existen muchos programas compatibles con mi SO. Eso tambien influye en que los ejetuables no sean tan faciles de encontrar, y aunque costo un poco de trabajo el ejecutar los programas al final se logro.

Fuentes de Informacion.

<https://mlir.llvm.org/>

<https://www.tensorflow.org/mlir?hl=es-419>

<https://es.wikipedia.org/wiki/LLVM>

<https://llvm.org/>

<https://www.europeanvalley.es/noticias/que-es-un-compiler-en-programacion/#:~:text=%C2%BFQu%C3%A9%20es%20un%20compilador%3F,lenguaje%20ensamblador%20al%20lenguaje%20m%C3%A1quina.>

<https://es.ryte.com/wiki/Compilador>

<https://blog.becas-santander.com/es/metodologias-desarrollo-software.html>

<https://rulonder.github.io/2016/05/04/fortran-debug-in-visual-studio-code/>

https://brew.sh/index_es

<https://stackoverflow.com/questions/33162757/how-to-install-gdb-debugger-in-mac-osx-el-capitan>

<https://github.com/llvm/llvm-project/tree/main/flang>

<https://llvm.org/>

<https://github.com/flang-compiler/flang>

<https://www.scivision.dev/flang-compiler-build-tips/>