UNAM
FACULTAD DE INGENIERIA

Compilador
Linkers

Project Manager
Hernández Hernández Cristian

System Architect
Bautista Chan Yasmin

System Tester
Laparra Miranda Sandra

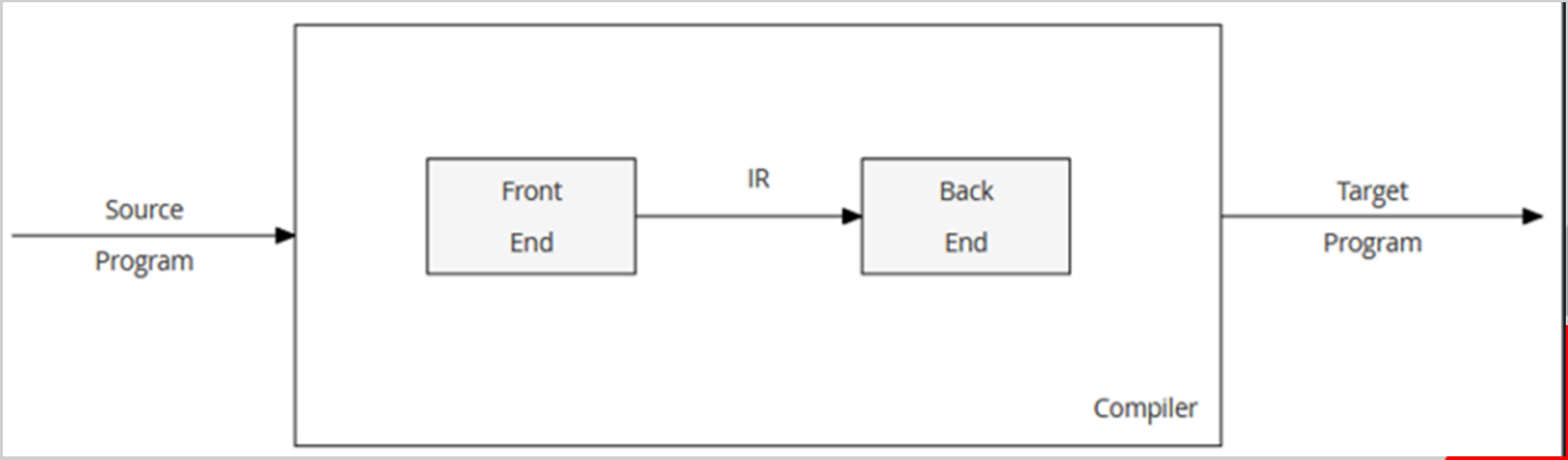
System Integrator
Terán Hernández Aldo

Introducción

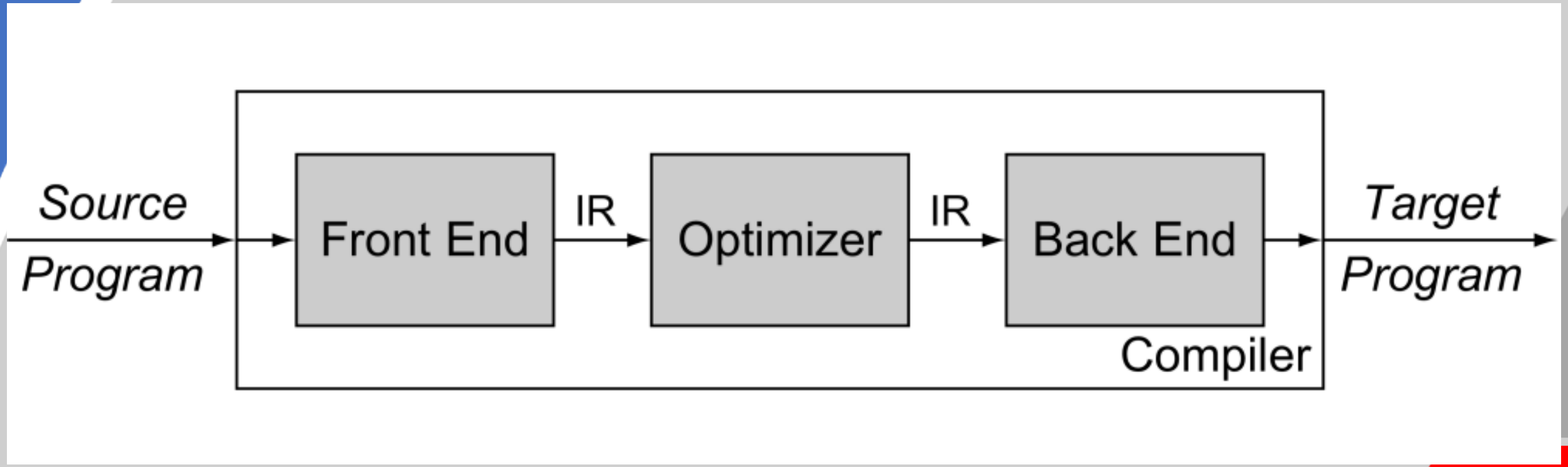
- Compilador
  - Es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar.

Componentes

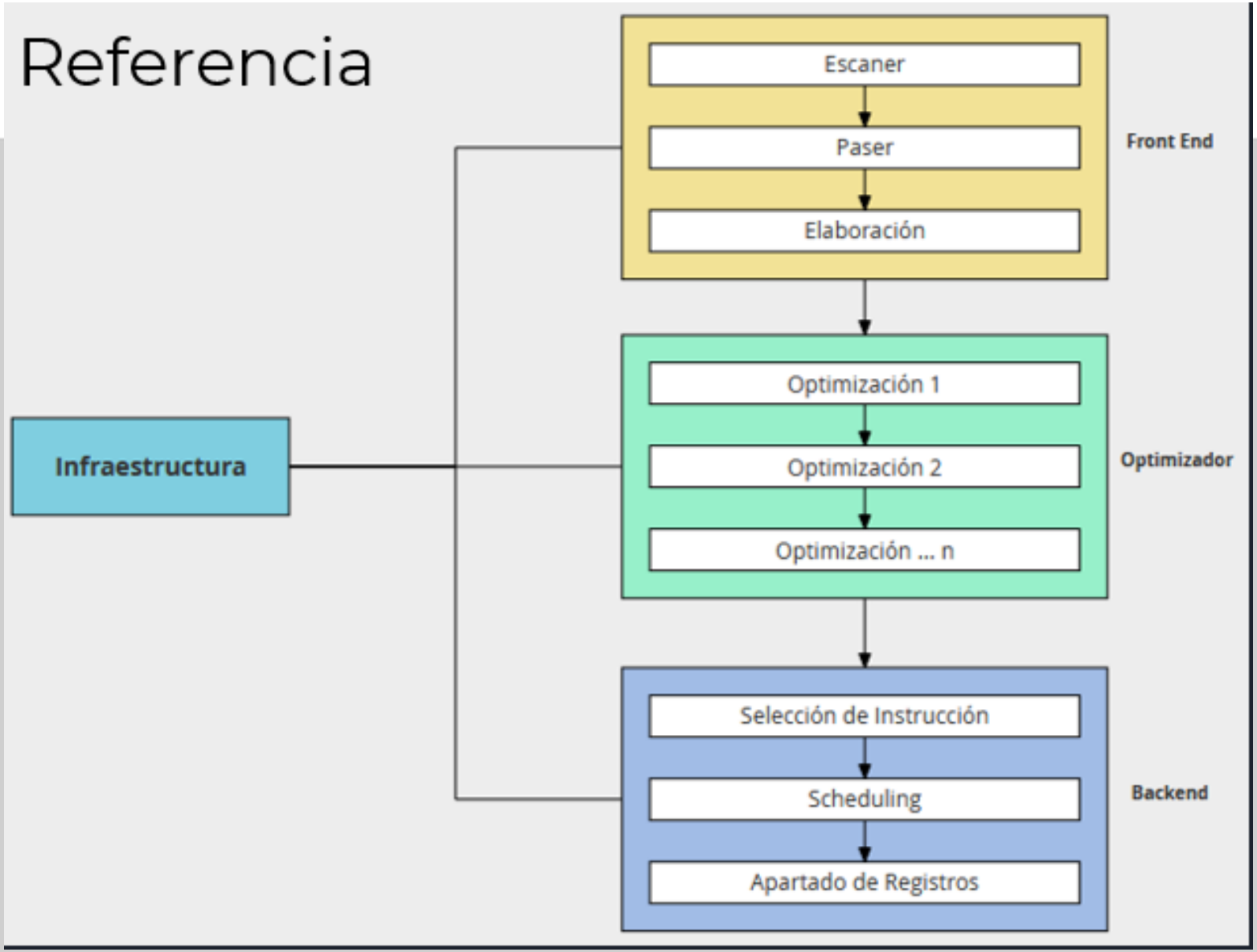
- Análisis léxico, Análisis sintáctico y Análisis semántico. Entre otras de las fases se encuentra la de síntesis que consiste en generar el código objeto equivalente al programa fuente.



Infraestructura

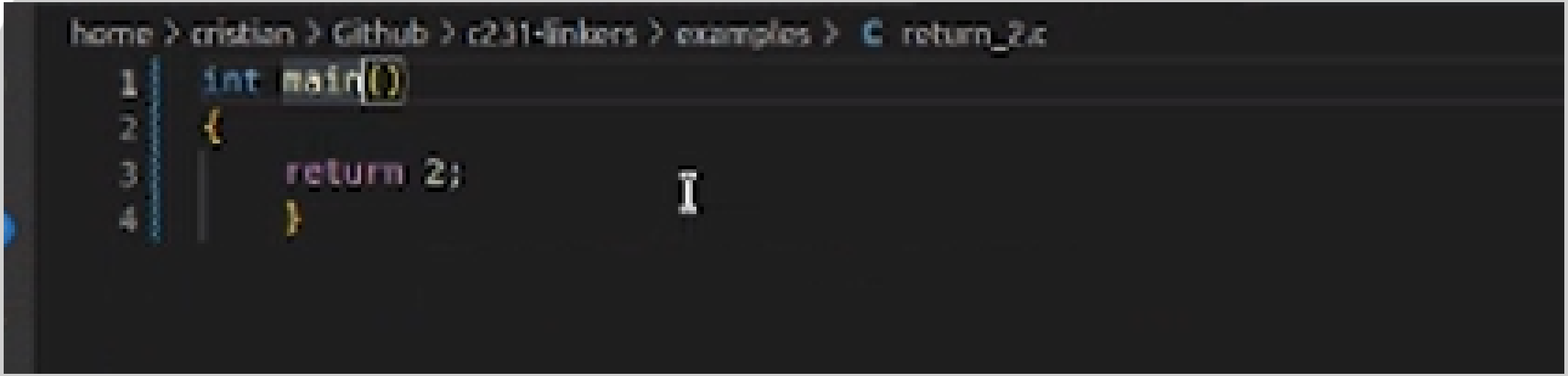


Referencia

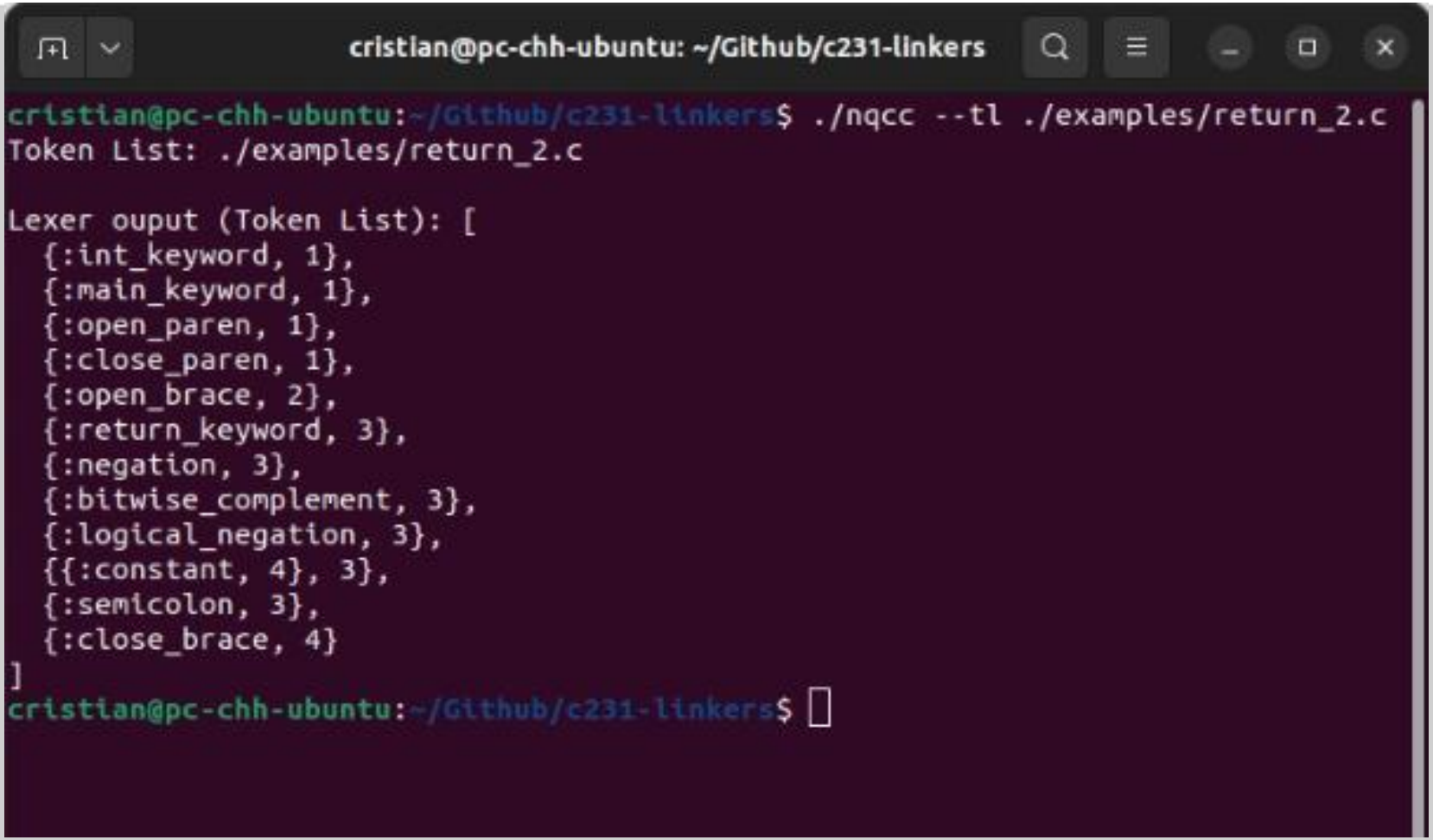

Escaner
Paser
Elaboración
Front End
Optimización 1
Optimización 2
Optimización ... n
Optimizador
Infraestructura
Selección de Instrucción
Scheduling
Apartado de Registros
Backend

Estructura Parte I

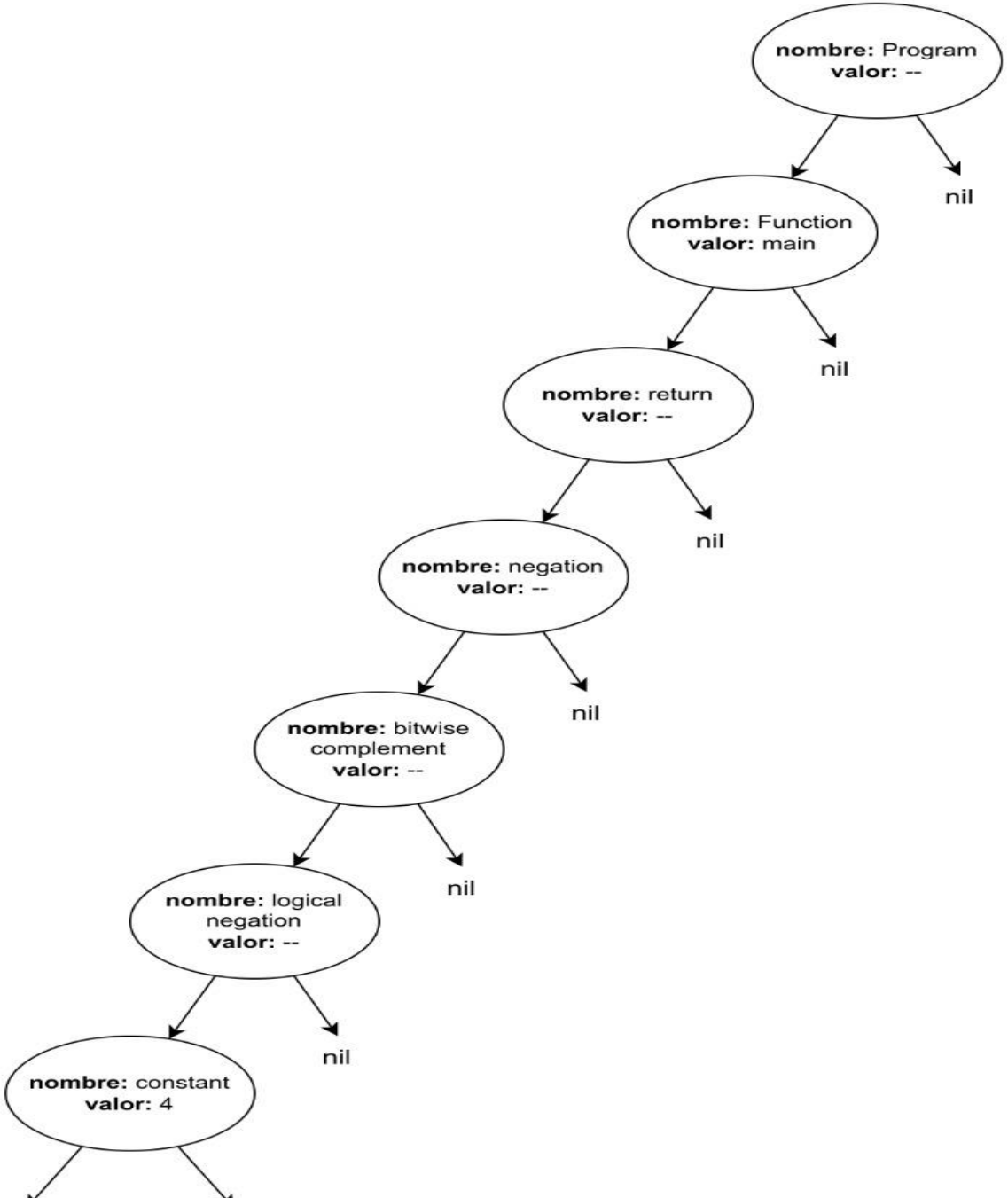
- Enteros
- Lexing
- Análisis sintáctico

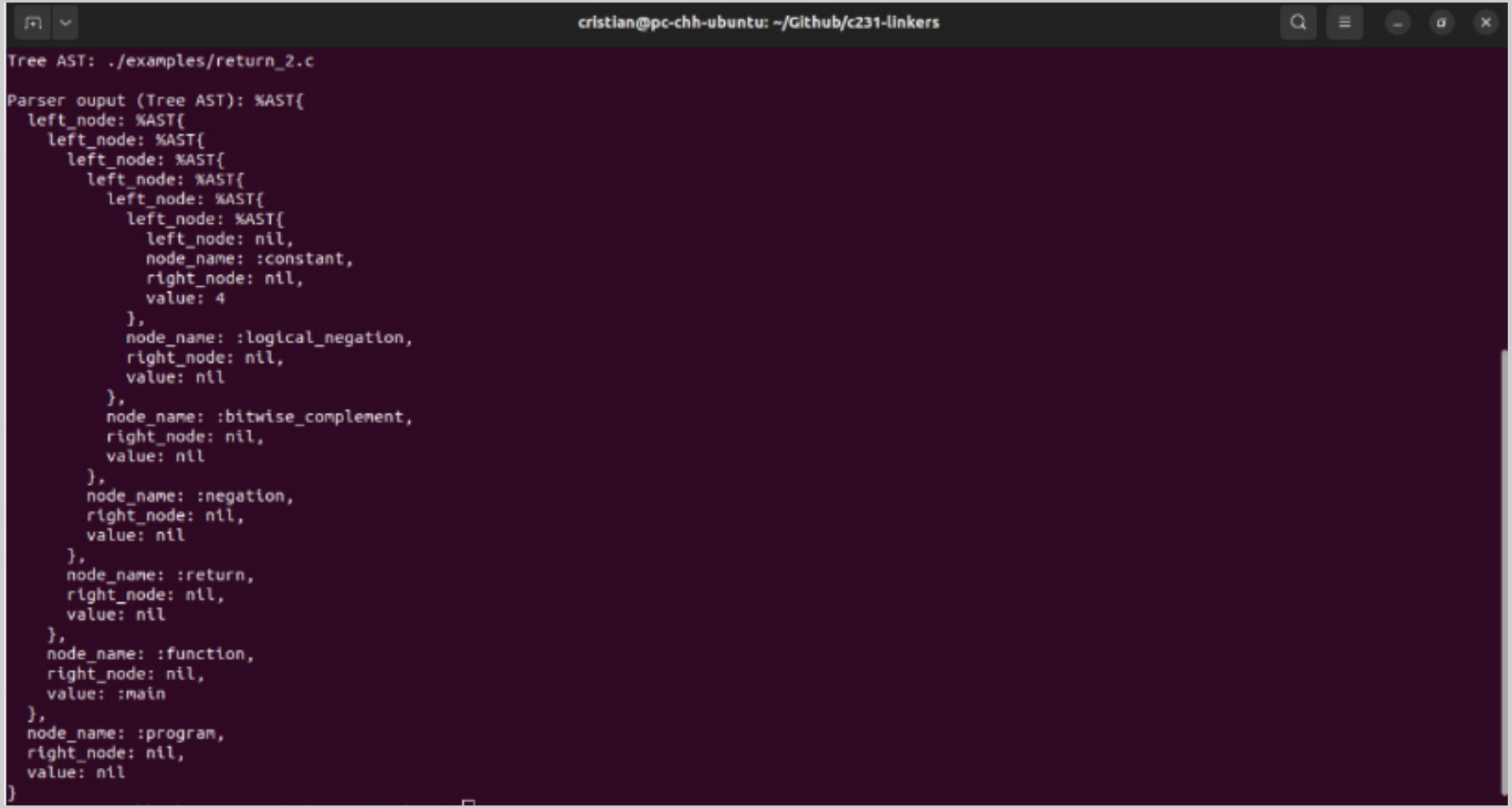

home > cristian > Github > c231-linkers > examples > return_2.c
1 int main()
2 {
3 return 2;
4 }

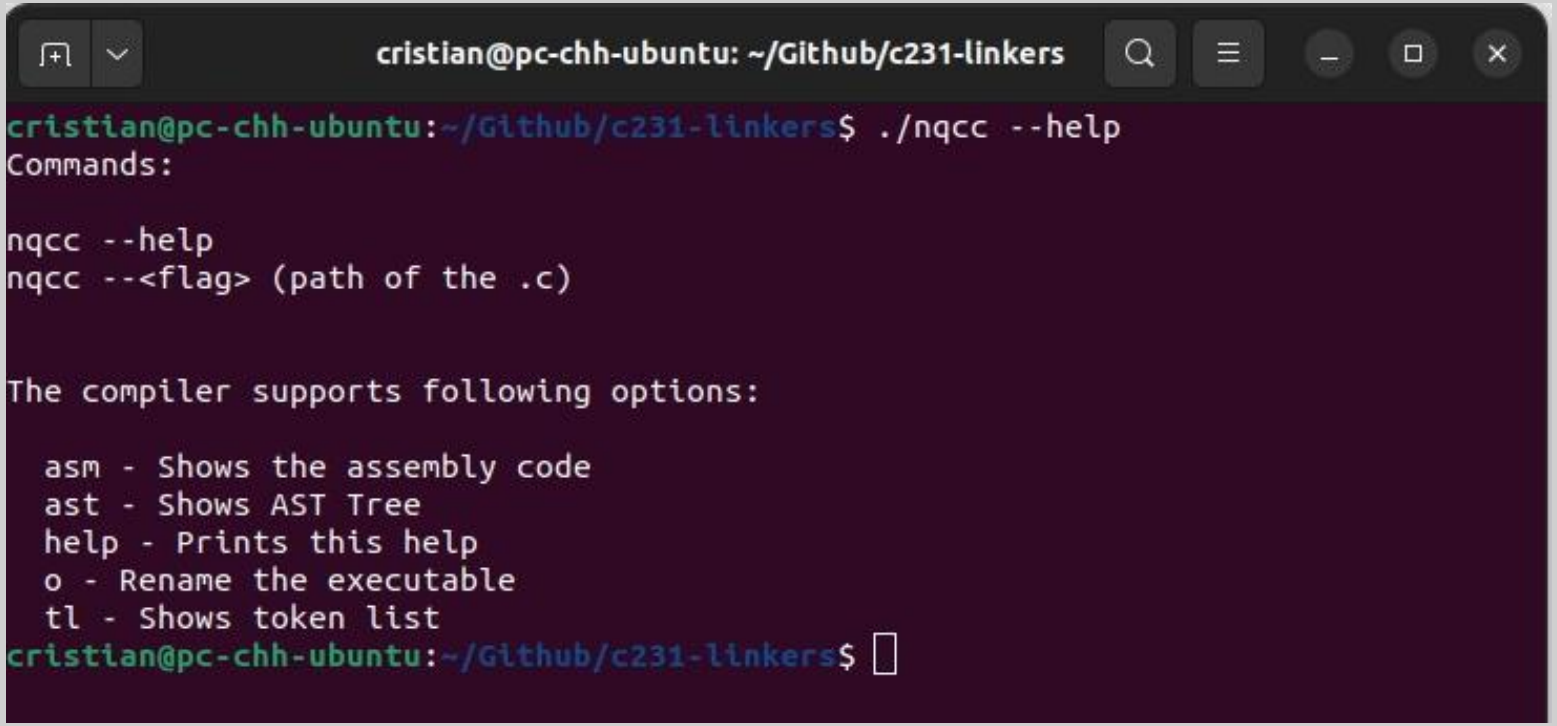
Lista de tokens

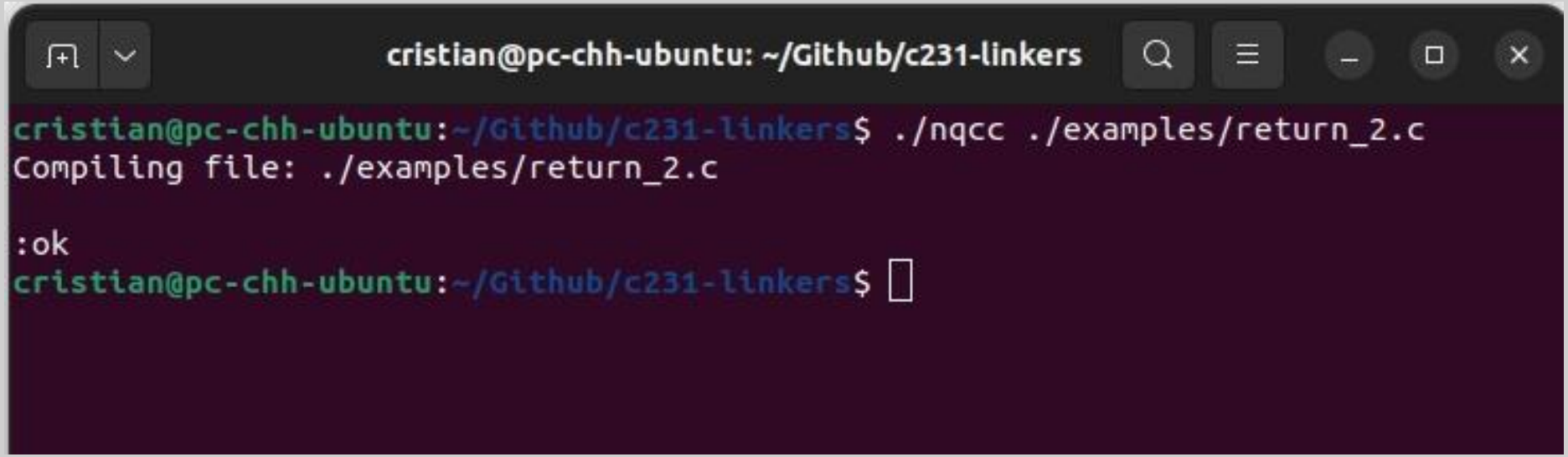


Árbol AST


nombre: Program
valor: --
nil
nombre: Function
valor: main
nil
nombre: return
valor: --
nil
nombre: negation
valor: --
nil
nombre: bitwise complement
valor: --
nil
nombre: logical negation
valor: --
nil
nombre: constant
valor: 4

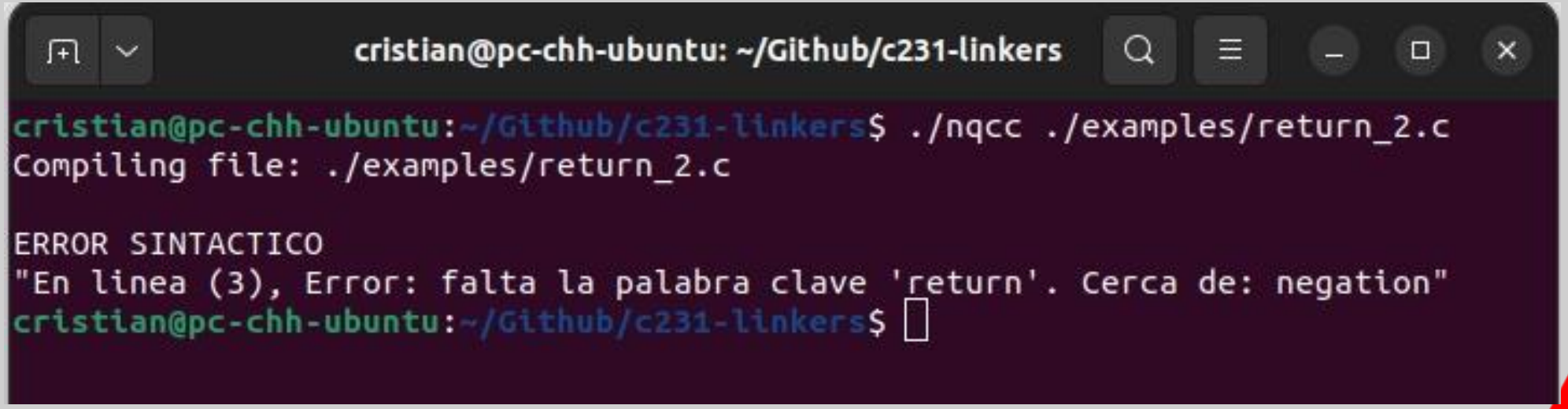

cristian@pc-chh-ubuntu: ~/Github/c231-linkers
Tree AST: ./examples/return_2.c
Parser ouput (Tree AST): %AST{
left_node: %AST{
left_node: %AST{
left_node: %AST{
left_node: %AST{
left_node: %AST{
left_node: %AST{
left_node: nil,
node_name: :constant,
right_node: nil,
value: 4
},
node_name: :logical_negation,
right_node: nil,
value: nil
},
node_name: :bitwise_complement,
right_node: nil,
value: nil
},
node_name: :negation,
right_node: nil,
value: nil
},
node_name: :return,
right_node: nil,
value: nil
},
node_name: :function,
right_node: nil,
value: :main
},
node_name: :program,
right_node: nil,
value: nil
}


cristian@pc-chh-ubuntu: ~/Github/c231-linkers
cristian@pc-chh-ubuntu:~/Github/c231-linkers$ ./nqcc --help
Commands:
nqcc --help
nqcc --<flag> (path of the .c)
The compiler supports following options:
asm - Shows the assembly code
ast - Shows AST Tree
help - Prints this help
o - Rename the executable
tl - Shows token list
cristian@pc-chh-ubuntu:~/Github/c231-linkers$


cristian@pc-chh-ubuntu: ~/Github/c231-linkers
cristian@pc-chh-ubuntu:~/Github/c231-linkers$ ./nqcc ./examples/return_2.c
Compiling file: ./examples/return_2.c
:ok
cristian@pc-chh-ubuntu:~/Github/c231-linkers$

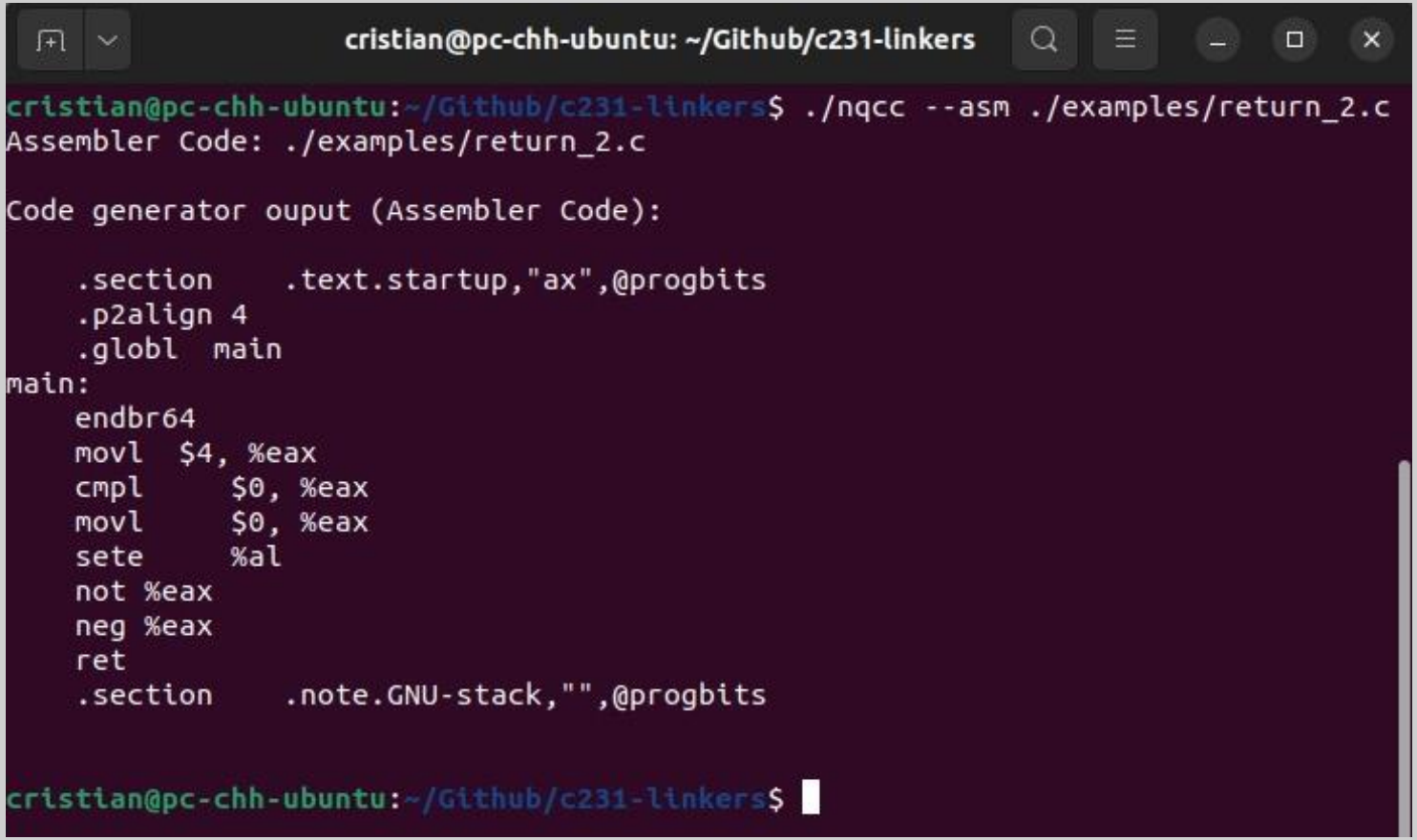

cristian@pc-chh-ubuntu: ~/Github/c231-linkers
cristian@pc-chh-ubuntu:~/Github/c231-linkers$ ./nqcc ./examples/return_2.c
Compiling file: ./examples/return_2.c
"Error lexico: (in) se encuentra un token no reconocido, en linea: 1"
"Error lexico: (retur) se encuentra un token no reconocido, en linea: 3"
cristian@pc-chh-ubuntu:~/Github/c231-linkers$

- En el caso de un error sintáctico, el mensaje muestra entre paréntesis la línea donde se encuentra el error, mientras que el mensaje descriptivo dependerá del tipo de falla detectado. Finalmente el "Cerca de: --" indica el token más cercano en donde esta el error.



Estructura Parte 2

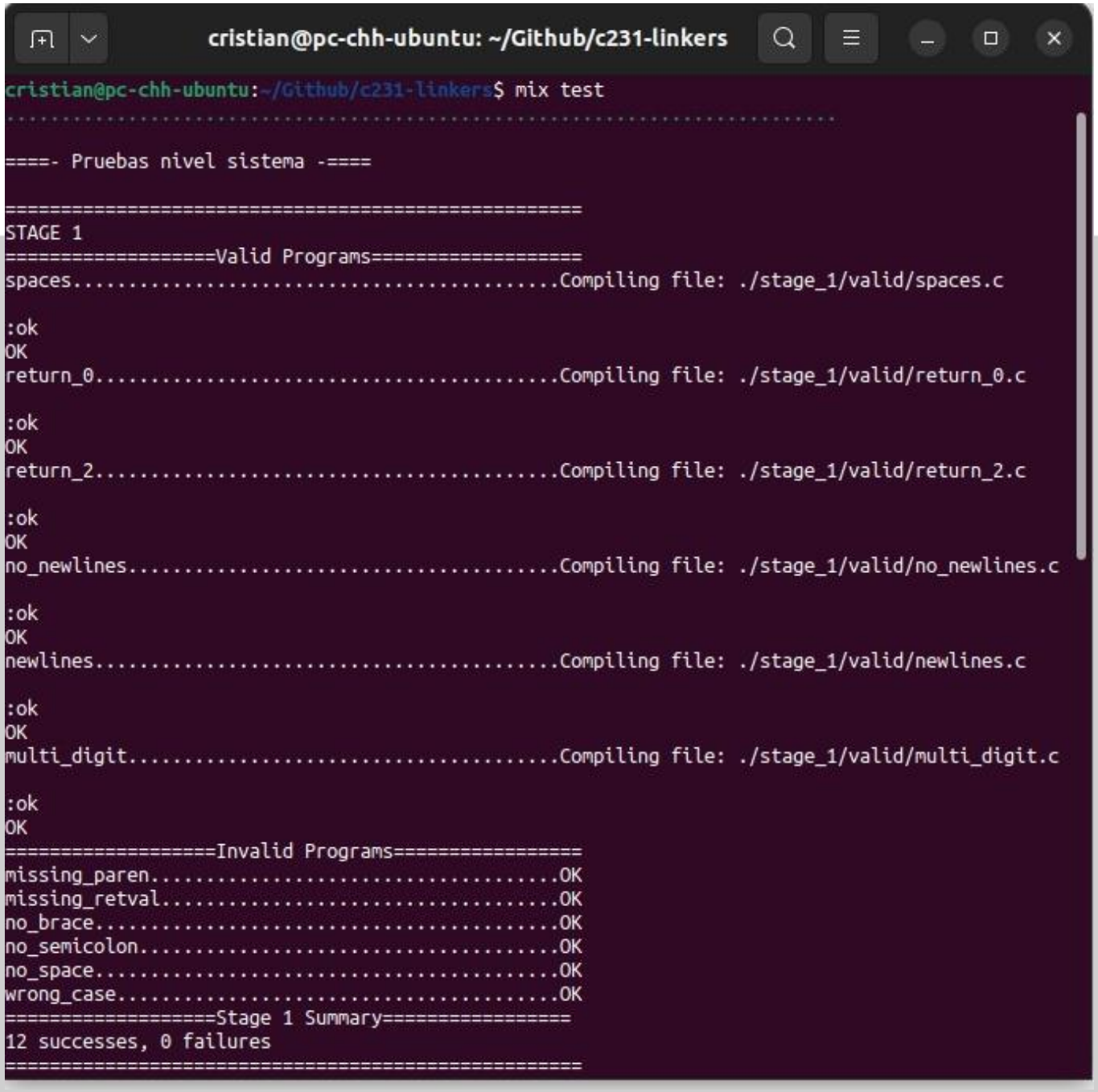
- Operadores Unarios
- Negación
- Complemento
- Negación Lógica


cristian@pc-chh-ubuntu: ~/Github/c231-linkers
cristian@pc-chh-ubuntu:~/Github/c231-linkers$ ./nqcc --asm ./examples/return_2.c
Assembler Code: ./examples/return_2.c
Code generator ouput (Assembler Code):
.section .text.startup,"ax",@progbits
.p2align 4
.globl main
main:
endbr64
movl $4, %eax
cmpl $0, %eax
movl $0, %eax
sete %al
not %eax
neg %eax
ret
.section .note.GNU-stack,"",@progbits
cristian@pc-chh-ubuntu:~/Github/c231-linkers$

```elixir
def lex_raw_tokens({program,linea}) when program != "" do
  code_line = linea
  {token, rest} =
    case program do
      "{" <> rest ->
        {{:open_brace, code_line}, rest}

      "}" <> rest ->
        {{:close_brace, code_line}, rest}

      "(" <> rest ->
        {{:open_paren, code_line}, rest}

      ")" <> rest ->
        {{:close_paren, code_line}, rest}

      ";" <> rest ->
        {{:semicolon, code_line}, rest}

      "-" <> rest ->
        {{:negation, code_line}, rest}

      "~" <> rest ->
        {{:bitwise_complement, code_line}, rest}

      "!" <> rest ->
        {{:logical_negation, code_line}, rest}

      "return" <> rest ->
        if String.first(rest) in ["{","}","(",")",";","-","~","!",nil] do
          {{:return_keyword, code_line}, rest}
```

```
      "return" <> rest ->
        if String.first(rest) in ["{","}","(",")",";","-","~","!",nil] do
          {{:return_keyword, code_line}, rest}
        else
          {:error,{"Token not valid: #{program}",linea}}
        end

      "int" <> rest ->
        if String.first(rest) in ["{","}","(",")",";","-","~","!",nil] do
          {{:int_keyword, code_line}, rest}
        else
          {:error,{"Token not valid: #{program}",linea}}
        end

      "main" <> rest ->
      if String.first(rest) in ["{","}","(",")",";","-","~","!",nil] do
        {{:main_keyword, code_line}, rest}
      else
        {:error,{"Token not valid: #{program}",linea}}
      end

      rest ->
        get_constant(rest,linea)
    end

  if token != :error do
    auxiliar_token={rest,linea}
    remaining_tokens=lex_raw_tokens(auxiliar_token)
    [token | remaining_tokens]
  else
    [{:error, rest}]
  end
end
```

Pruebas


cristian@pc-chh-ubuntu: ~/Github/c231-linkers
cristian@pc-chh-ubuntu:~/Github/c231-linkers$ mix test
.....
====- Pruebas nivel sistema -====
==========
STAGE 1
==========Valid Programs==========
spaces.....Compiling file: ./stage_1/valid/spaces.c
:ok
OK
return_0.....Compiling file: ./stage_1/valid/return_0.c
:ok
OK
return_2.....Compiling file: ./stage_1/valid/return_2.c
:ok
OK
no_newlines.....Compiling file: ./stage_1/valid/no_newlines.c
:ok
OK
newlines.....Compiling file: ./stage_1/valid/newlines.c
:ok
OK
multi_digit.....Compiling file: ./stage_1/valid/multi_digit.c
:ok
OK
==========Invalid Programs==========
missing_paren.....OK
missing_retval.....OK
no_brace.....OK
no_semicolon.....OK
no_space.....OK
wrong_case.....OK
==========Stage 1 Summary==========
12 successes, 0 failures
==========

Conclusiones

- Crear un compilador conlleva tener varios antecedentes sobre otros temas revisadas en anteriores cursos. Realizar el compilador nos hizo comprender de manera práctica su construcción y que elementos necesita para funcionar. Durante su desarrollo nos encontramos con varias dificultades como lo fue en un principio el nuevo lenguaje de programación al cambiar el paradigma a uno funcional, otra de las dificultades fue que los integrantes del equipo no coincidíamos varias veces con los horarios para realizar reuniones y avanzar con las entregas. Por otra parte, una vez que ya dominados un poco mas el lenguaje de programación la implementación de los cambios ya no fue una tarea de mayor complejidad. Cabe mencionar que el desempeño en general del equipo no es perfecta como se plantea desde un principio puesto que afectan varios factores externos.

Project Manager
Hernández Hernández Cristian

- De las cosas menos fáciles es aprender algo nuevo, y tratar de entenderlas es un reto mayor, si bien en esta oportunidad primero hay que darse el tiempo y acudir con las personas responsables para congeniar, negociar y distribuir actividades. En esta primera y segunda entrega podemos apreciar que no era una actividad fácil, puesto que primera instancia debíamos conocer las especificaciones del usuario general sobre las que se tendrían que investigar organizar y reordenar para llegar a un adecuado producto.
- Conociendo las especificaciones de nuestro usuario pues la comunicación del equipo desarrollador debería de ser clara dinámica y objetiva dando como resultado reuniones especificando que tiene que hacer cada miembro y como se debería de llevar a cabo; de las dificultades presentadas era el conocimiento sobre un nuevo lenguaje de programación la cual con orientación del usuario pudimos tener herramientas necesarias para dar comienzo al proyecto.

System Architect
Bautista Chan Yasmin

- Como comienzo del proyecto, pues analizamos las pautas dadas por el usuario en la cual se leyó analizo apuntes por parte de la ciudadana Nora Sandle la cual se a especializado en recrear compiladores de una manera más dinámica y efectiva en diferentes lenguajes de programación.
- Al estar revisando los apuntes pues se trabajo en la primera entrega que era demostrar la lectura de un entero la cual en un principio genero percances ya que no todos estábamos con los mismos conocimientos, pero posteriormente bajo la buena comunicación y responsabilidad se logró concluir este apartado permitiendo que se generara la segunda entrega la cual se facilito puesto se recabo mas información durante las sesiones con el usuario y la aclaración de cierto puntos que se daban por hecho; los puntos relevantes que se trabajaron internamente en el equipo son la lista de tokens, la representación del árbol así como la organización y distribución de cada uno de los apartador del compilador
- El trabajo en el equipo a sido dinámica, congruente y con comunicación asertiva, en el proceso si se a tenido inconvenientes de tiempo pero se han podido resolver de manera oportuna dando como resultado la presentación del trabajo realizo y planificando las siguientes entregas.

System Architect

Bautista Chan Yasmin

- En la primera entrega de este proyecto aprendí a diseñar e implementar un compilador para el lenguaje C y así poder compilar un programa que devuelve números enteros, mientras que en la segunda entrega actualizamos cada una de las etapas del compilador para poder aplicar operadores unarios a dichos números enteros. Cabe destacar que durante el proceso de construcción del compilador aprendí a implementar cada una de sus fases y así el compilador pueda analizar un programa escrito en un lenguaje de alto nivel que en nuestro caso es el lenguaje C, de tal forma que si es correcto genera un código equivalente escrito en otro lenguaje tal como ensamblador. Por otra parte examinamos el proceso de compilación para ver que opera como una secuencia de fases, cada una de las cuales transforma una representación del programa fuente en otro.

- La parte más sencilla de implementar fue la segunda entrega, ya que una vez que logramos entender y completar la primera entrega, seguimos los mismos pasos para agregar los nuevos tokens en el *lexer*, mientras que para la parte del *Parser* agregamos otro tipo de expresión que permitiera tomar ya sea una operación constante o unaria. Sin embargo, implementar el generador de código para la segunda entrega se complicó un poco, debido a que tuvimos que pensar en nuevas instrucciones.

System Tester
Laparra Miranda Sandra

- Una de las dificultades de realizar estas dos entregas fue aprender un nuevo lenguaje de programación, ya que el poco dominio de este lenguaje me limitó a implementar las instrucciones del algoritmo que ya teníamos armado, no obstante durante el proceso aprendí lo básico para poder realizar cada una de las fases del compilador. Otra de las dificultades fue entender cómo transformar la lista de tokens en la representación de un árbol AST, porque el proceso de programarlo no fue tan sencillo como realizarlo a mano.
- En cuanto al trabajo en equipo puedo comentar que logramos realizar las dos entregas sin tener ningún problema en organización y en tiempo, de manera que coordinamos todas nuestra actividades para alcanzar nuestros objetivos.

System Tester

Laparra Miranda Sandra

- El uso de elixir es algo complejo de primera mano debido a que es un paradigma nuevo para nosotros y es complicado su primer uso , sin embargo una vez aprendiendo las bases del lenguaje de buena forma y sobre todo la forma en que este funciona , ya todo se va dando de manera mas simple y entendible. Aprendimos primeramente el lenguaje elixir o al menos parte de el , además ya en el caso de la materia comprendimos como el compilador funciona en este caso como es que es la producción de el código que queremos generar pero para ello ocupamos un proceso metódico y algo largo en el sentido de que cada una de las piezas que conforma el compilador es escencial para que este funcione de manera correcta. Como cada una de estas piezas , el parser , lexer , code generator, linker y el sanitzer, nos ayudan cada uno a conformar poco a poco el código.

System Integrator

Terán Hernández Aldo

- La dificultad que se tiene con todo esto fue entender primeramente la estructura de un compilador y aplicarlo , una vez entendido eso y el lenguaje todo fluye de manera mas natural , sin embargo si fue muy complejo al principio entender todo el funcionamiento.El tiempo fue organizado de manera en que se cubriera todo de buena manera , una organizacion buena junto con una comunicacion muy eficientepara que todo salga como debe de salir , de manera correcta y sin contratiempos.

Apéndice

- https://github.com/hiphoox/c231-linkers

Bibliografía

- *Nora Sandler*. (2017, 29 noviembre). https://norasandler.com/2017/11/29/Write-aCompiler.html