



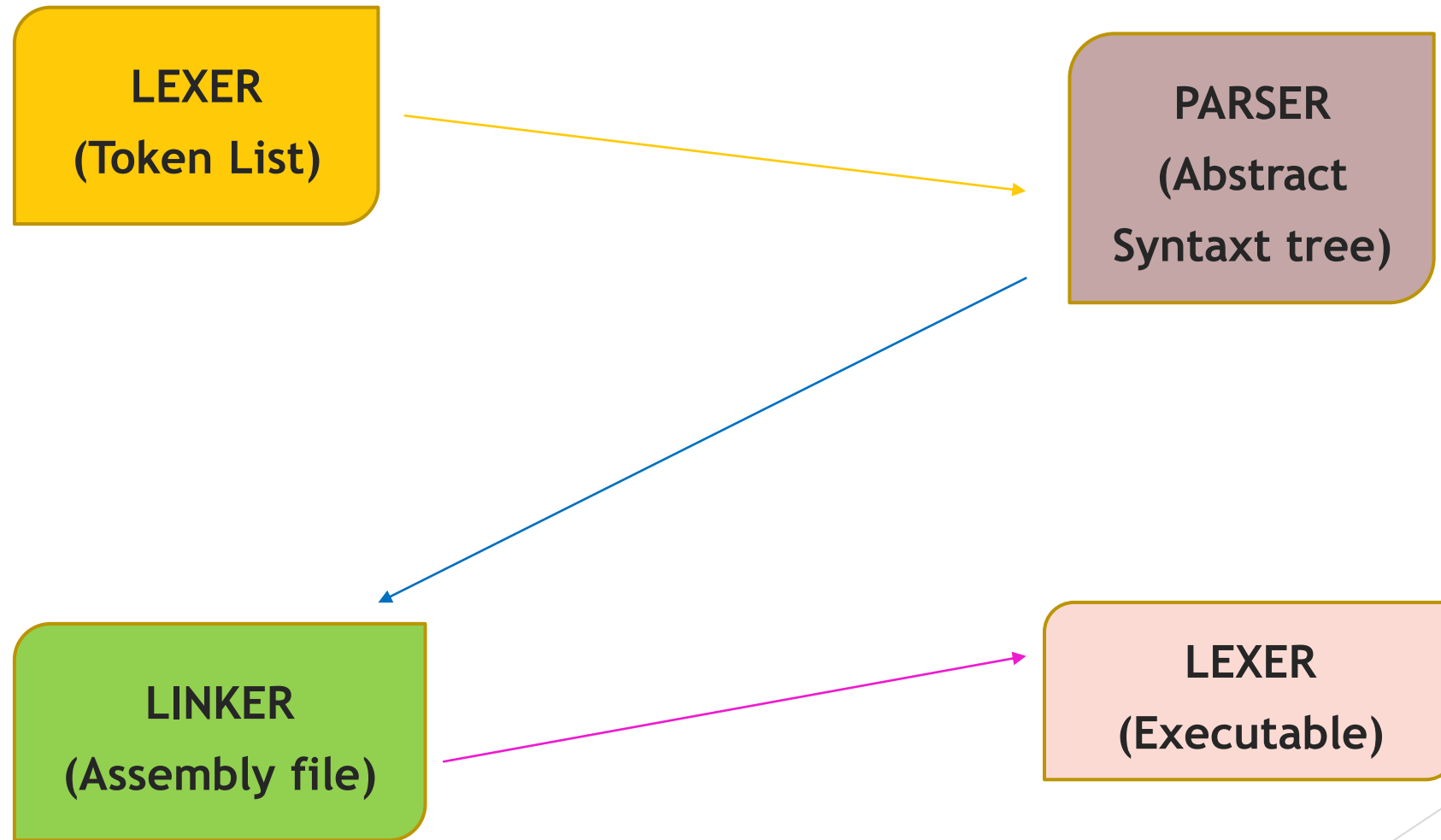
Gold Compilers: Compiler “Foxy”

Second stage

Integrants:

- ❖ García Felipe Miguel (Project Manager)
- ❖ Felix Flores Paul Jaime (Tester)
- ❖ SanJuan Aldape Diana Paola (The System Integrator)

How is our compiler composed?



Lexer:

This integration will validate this:

- ✚ Validate that list of tokens.
- ✚ The output will be a list of atom strings tuples.
- ✚ If there is an error, a list of tuples with the token will be displayed, as well as the wrong column and row.

Parser:

- ▶ This integration will allow us to establish the following basic functionality:
- ▶ Generate an AST with the list of tuples created by the Lexer.
- ▶ If there is some error, it will display a list of tuples with the token generating the error, the column, and the row.

Code generator:

- ▶ This integration will allow us to establish the following basic functionality:
- ▶ Take the AST generated by the Parser to build the code in assembler, from the leaves to the root.
- ▶ The output will be a string with the representative code in assembler.

Linker:

- ▶ Linker: is a computer System program that takes one or more object files generated by a compiler or an assembler and combines them into a single executable file, library file, or another 'object' file

The background features a dark grey field on the left and a series of overlapping, semi-transparent yellow and orange geometric shapes on the right. These shapes create a dynamic, layered effect. A thin white line runs diagonally across the right side, passing through the overlapping shapes.

Stage one:

First delivery:

- ▶ Compile a C source code and return a integer when executes de .exe file.
- ▶ To achieve this objective, we will base ourselves on how to create a compiler according to Nora's documentation, as well as the tests that must be executed.

First Implentation:

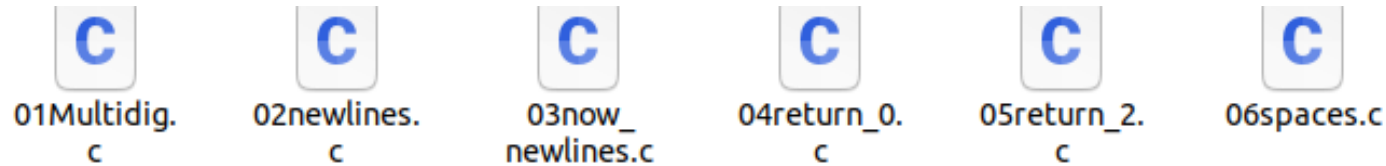
```
tokens = [
    { :type, :intKeyword},#      int
    { :ident, :returnKeyword},#  return
    { :ident, :mainKeyword},#    main
    { :left_brace},#             {
    { :right_brace},#            }
    { :left_paren},#             (
    { :right_paren},#            )
    { :semicolon},#             ;
    ##### Segunda entrega #####
```

Test that the compiler must to do:

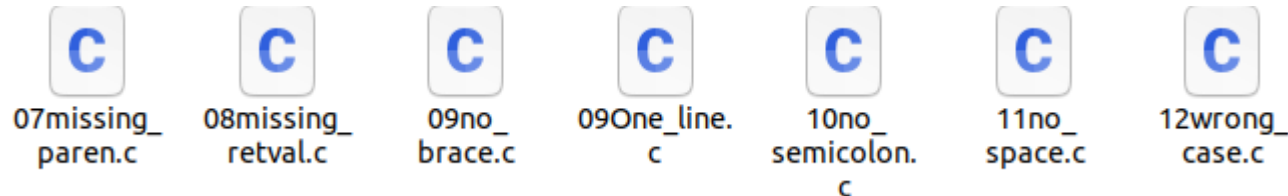
- ▶ You can compile several tests that we have, if you want it in the same way, but here we will only do one

You can see the examples valid e invalids in the folders:

Valids :



Invalids:



Test of Nora first stage:

https://github.com/nlsandler/write_a_c_compiler/tree/master/stage_1

► Valid:

multi_digit.c
newlines.c
no_newlines.c
return_0.c
return_2.c
spaces.c

► Invalid:

missing_paren.c
missing_retval.c
no_brace.c
no_semicolon.c
no_space.c
wrong_case.c

Example stage one:

Valid example:

```
int main(){  
    return 2;  
}
```

Invalid example:

```
int main(){  
    return 2  
}
```

How to use the compiler Foxy with:

```
parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/GoldCompilers/CompiladorFoxy$ ./foxy -h

*****

You can use the following shortcuts:

    -t [filename.c]  It shows us the token list.
    -a [filename.c]  It shows us the AST.
    -s [filename.c]  It shows us the assembly.
    -c [filename.c]  It compile program.

*****

parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/GoldCompilers/CompiladorFoxy$
```

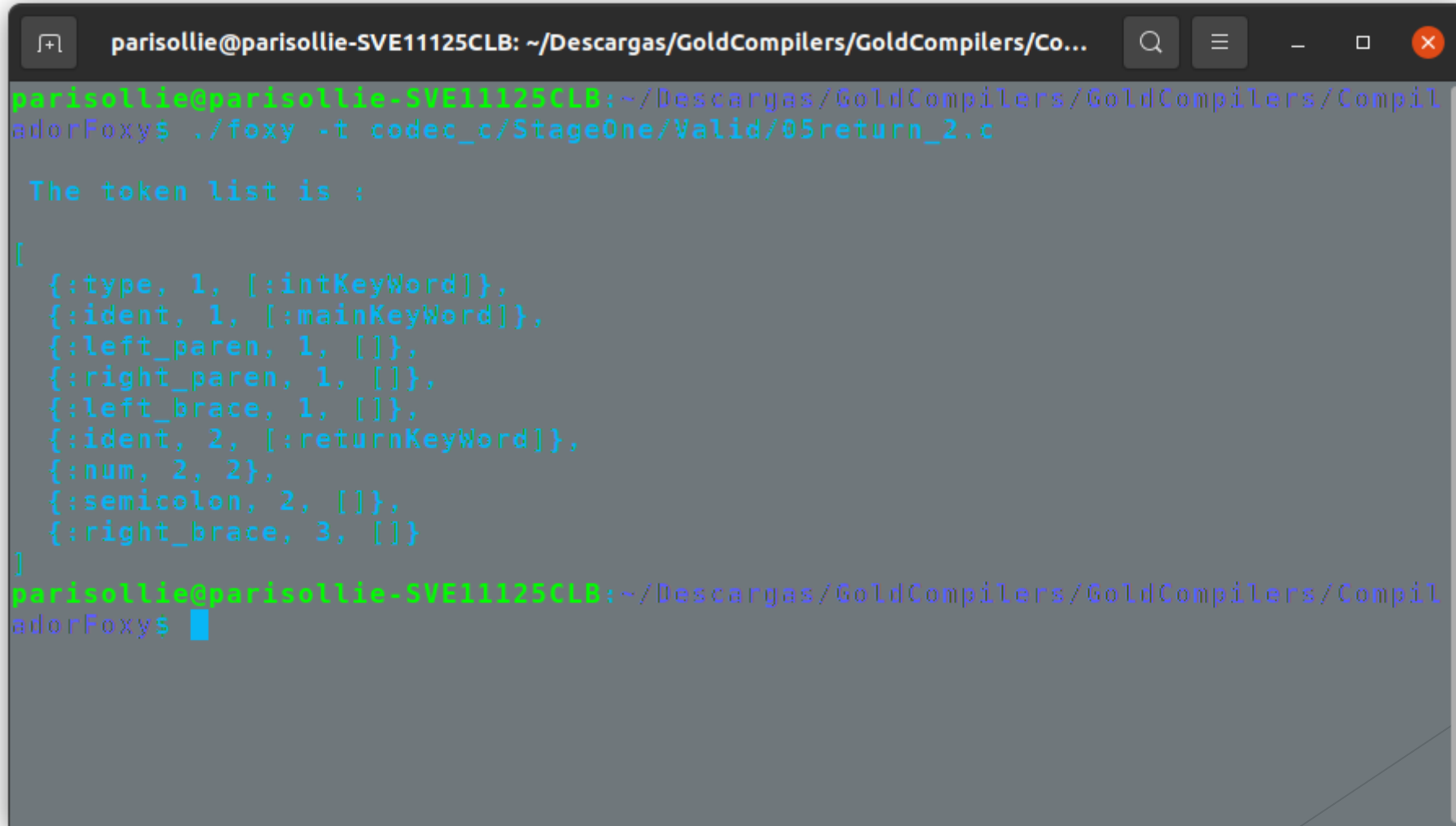
(-a)Development language must be a matching pattern to easily build an Abstract Syntax Tree (AST), however, phase I the right side's tree must be nil.

```
parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/Co...
adorFoxys$ ./foxy -a codec_c/StageOne/Valid/05return_2.c

The AST is:

%AST{
  lf_node: %AST{
    lf_node: %AST{
      lf_node: %AST{lf_node: nil, node_name: :constant, rt_node: nil, val: 2},
      node_name: :return,
      rt_node: nil,
      val: :return
    },
    node_name: :function,
    rt_node: nil,
    val: :main
  },
  node_name: :program,
  rt_node: nil,
  val: nil
}
parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/Compil
adorFoxys$
```

(-t) We show token's list from source code. Must check a relational couple to recognize every token.

A terminal window with a dark background and light-colored text. The window title is 'parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/Co...'. The prompt is 'parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/CompiladorFoxy\$'. The command entered is './foxy -t codec_c/StageOne/Valid/05return_2.c'. The output is 'The token list is :' followed by a list of tokens in a structured format. The tokens are: {type, 1, [:intKeyword]}, {ident, 1, [:mainKeyword]}, {left_paren, 1, []}, {right_paren, 1, []}, {left_brace, 1, []}, {ident, 2, [:returnKeyword]}, {num, 2, 2}, {semicolon, 2, []}, and {right_brace, 3, []}.

```
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/CompiladorFoxy$ ./foxy -t codec_c/StageOne/Valid/05return_2.c

The token list is :

[
  {type, 1, [:intKeyword]},
  {ident, 1, [:mainKeyword]},
  {left_paren, 1, []},
  {right_paren, 1, []},
  {left_brace, 1, []},
  {ident, 2, [:returnKeyword]},
  {num, 2, 2},
  {semicolon, 2, []},
  {right_brace, 3, []}
]
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/CompiladorFoxy$
```


(-s)Assembly must write in 64-bits set instructions.
The assembly syntax must be a AT and T by default in GCC.

```
parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/Co...
adorFoxys$ ./foxy -s codec_c/StageOne/Valid/05return_2.c

The assembly code is :

.section          __TEXT,__text,regular,nat_Instructs
.p2align          4, 0x90
.globl _main      ## Begin function main
_main:            ## @main
movl    2, %eax
push    %rax
pop     %rbx
ret
push    %rax
pop     %rbx
push    %rax
pop     %rbx

parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/Compil
adorFoxys$
```

(-c) We get the assembler and the executable:

```
parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/GoldCompilers/C...
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/GoldCompilers/C
ompiladorFoxys$ ./foxy -c codec_c/StageOne/Valid/05return_2.c

*****

The compiling the file is :
codec_c/StageOne/Valid/05return_2.c

*****

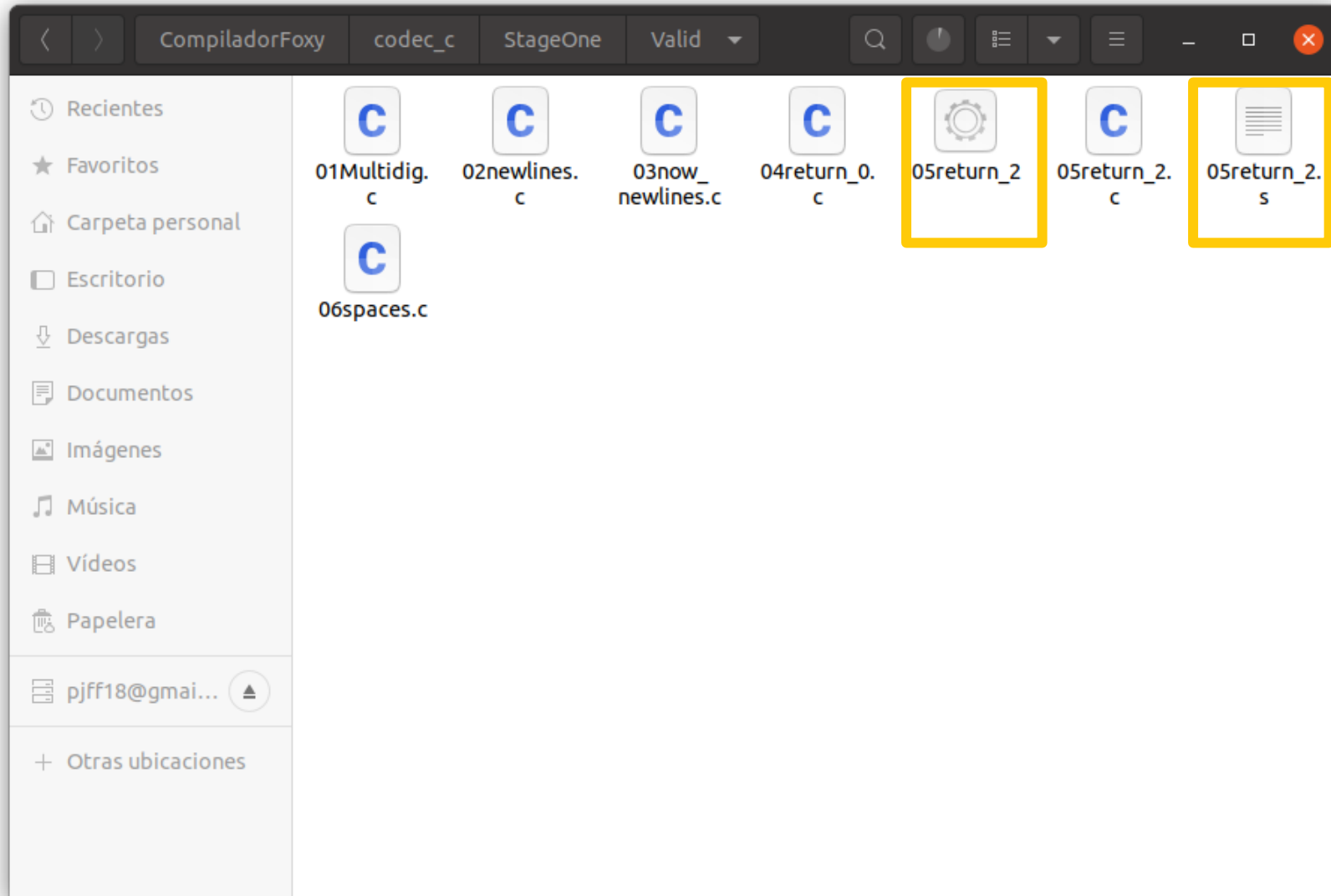
The program has worked correctly :D, your executables have been generated:
.....

The assembler it's located in the following route: codec_c/StageOne/Valid/05return_2.s
.....

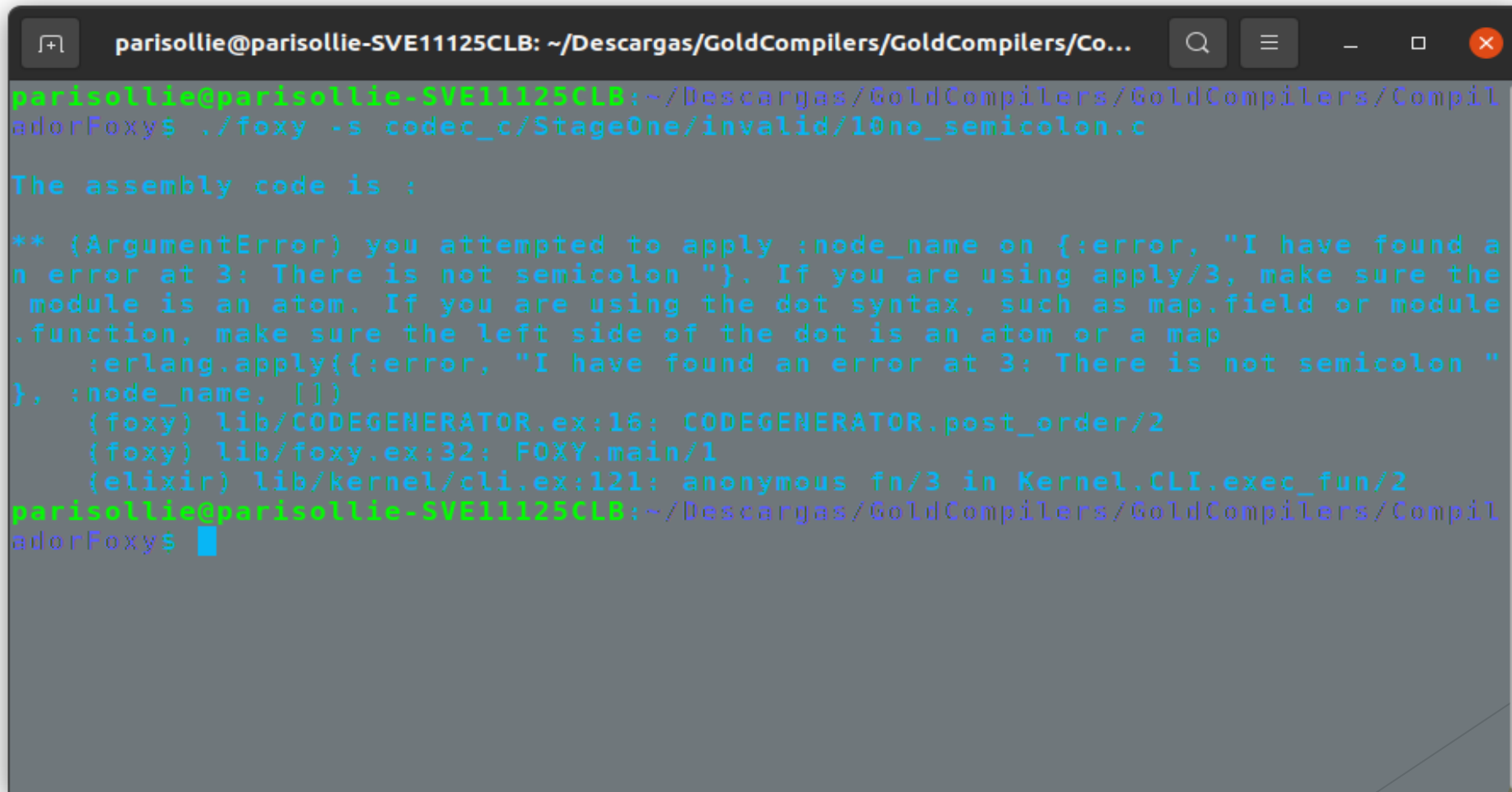
The executable it's located in the following route: codec_c/StageOne/Valid/./05return_2
.....

parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/GoldCompilers/C
ompiladorFoxys$
```

It is generated in our folder:



Test Invalid:

A terminal window with a dark background and light-colored text. The window title is "parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/Co...". The prompt is "parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/CompilerFoxy\$". The command entered is "./foxy -s codec_c/StageOne/invalid/10no_semicolon.c". The output shows the assembly code and a detailed error message: "** (ArgumentError) you attempted to apply :node_name on {:error, \"I have found an error at 3: There is not semicolon \"}. If you are using apply/3, make sure the module is an atom. If you are using the dot syntax, such as map.field or module.function, make sure the left side of the dot is an atom or a map\". :erlang.apply({:error, \"I have found an error at 3: There is not semicolon \"}, :node_name, []). (foxy) lib/CODEGENERATOR.ex:16: CODEGENERATOR.post_order/2 (foxy) lib/foxy.ex:32: FOXY.main/1 (elixir) lib/kernel/cli.ex:121: anonymous fn/3 in Kernel.CLI.exec_fun/2". The prompt returns to "parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/CompilerFoxy\$".

```
parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/Co...
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/CompilerFoxy$ ./foxy -s codec_c/StageOne/invalid/10no_semicolon.c

The assembly code is :

** (ArgumentError) you attempted to apply :node_name on {:error, "I have found an
error at 3: There is not semicolon "}. If you are using apply/3, make sure the
module is an atom. If you are using the dot syntax, such as map.field or module
.function, make sure the left side of the dot is an atom or a map
:erlang.apply({:error, "I have found an error at 3: There is not semicolon "
}, :node_name, [])
(foxy) lib/CODEGENERATOR.ex:16: CODEGENERATOR.post_order/2
(foxy) lib/foxy.ex:32: FOXY.main/1
(elixir) lib/kernel/cli.ex:121: anonymous fn/3 in Kernel.CLI.exec_fun/2
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/CompilerFoxy$
```

The background features a dark grey field on the left and a series of overlapping, semi-transparent yellow and orange geometric shapes on the right. These shapes include triangles and polygons of various sizes and orientations, creating a dynamic, layered effect. A thin white line runs diagonally across the right side, intersecting the yellow shapes.

Stage two:

Second delivery:

- ▶ Our compiler must can return a result operated with unary operations like negate result, positive, bitwise, and logical negation.
- ▶ To achieve this objective, we will base ourselves on how to create a compiler according to Nora's documentation, as well as the tests that must be executed.

Second Implentation:

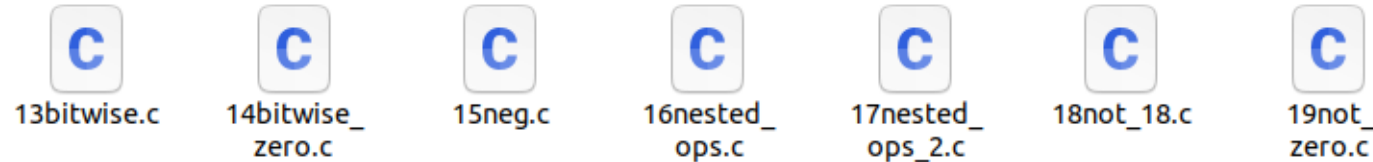
```
##### Segunda entrega #####  
{:op, :neg},# -  
{:op, :log_Neg},# !  
{:op, :bW},# ~  
##### Tercera entrega #####
```

Test that the compiler must to do:

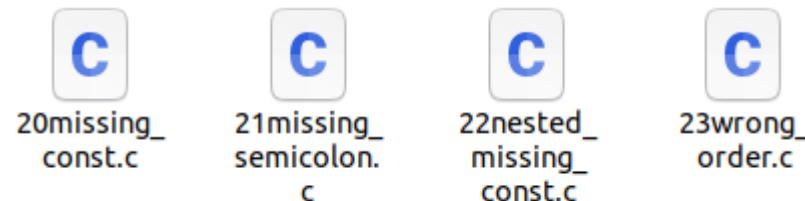
- ▶ You can compile several tests that we have, if you want it in the same way, but here we will only do one

You can see the examples valid e invalids in the folders:

Valids :



Invalids:



Test of Nora second stage:

https://github.com/nlsandler/write_a_c_compiler/tree/master/stage_2

► Valid:

- bitwise.c
- bitwise_zero.c
- neg.c
- nested_ops.c
- nested_ops_2.c
- not_five.c
- not_zero.c

► Invalid:

- missing_const.c
- missing_semicolon.c
- nested_missing_const.c
- wrong_order.c

Example stage two:

Valid example:

```
int main(){  
    return !17;  
}
```

Invalid example:

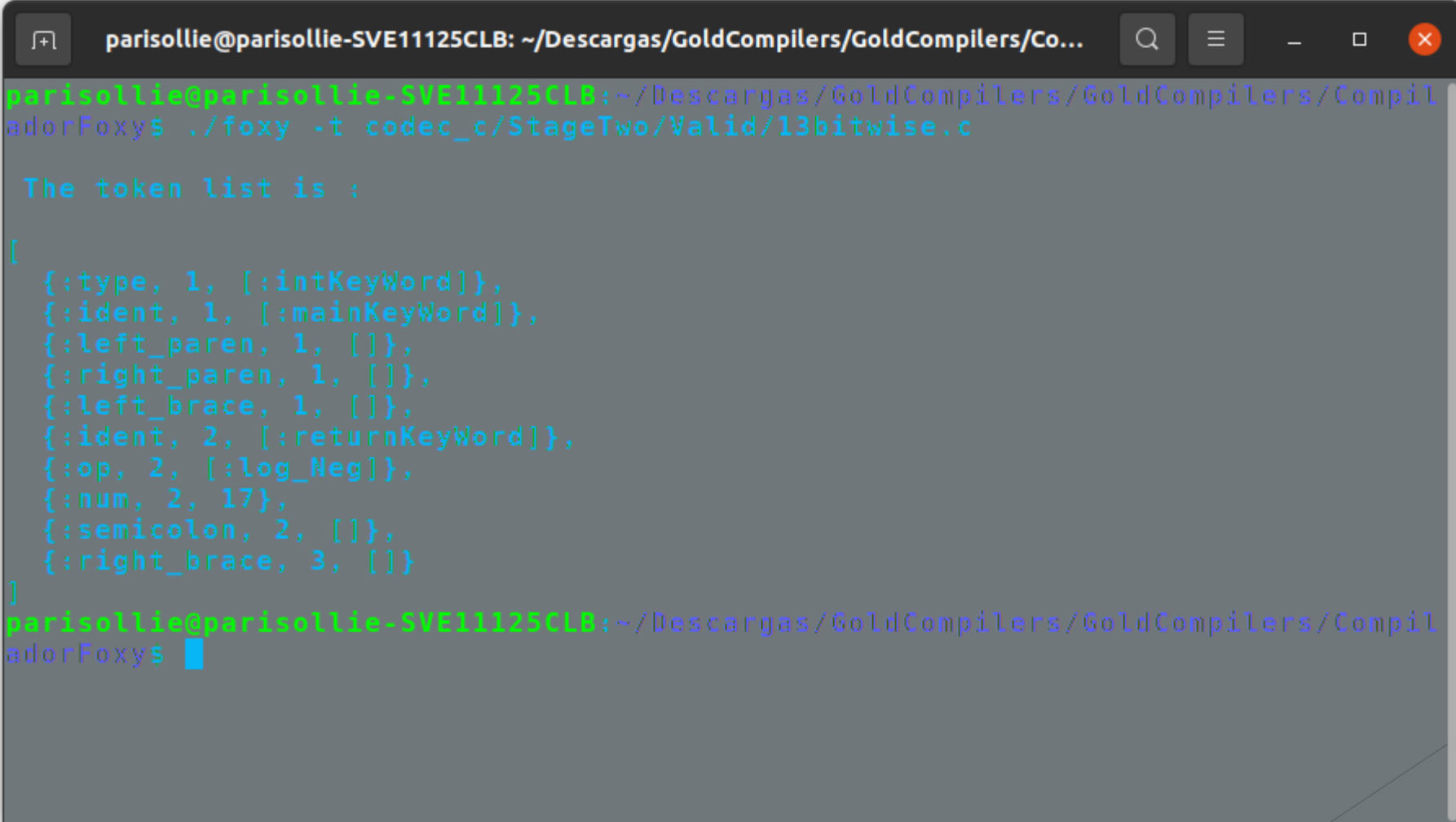
```
int main(){  
    return !;  
}
```

(-a) Development language must be a matching pattern to easily build an Abstract Syntax Tree (AST), however, phase I the right side's tree must be nil.

```
parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/CompiladorFoxy
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/CompiladorFoxy$ ./foxy -a codec_c/StageTwo/Valid/13bitwise.c

The AST is:
%AST{
  lf_node: %AST{
    lf_node: %AST{
      lf_node: %AST{
        lf_node: nil, node_name: :constant, rt_node: nil, val: 17},
        node_name: :unary,
        rt_node: nil,
        val: :log_Neg
      },
      node_name: :return,
      rt_node: nil,
      val: :return
    },
    node_name: :function,
    rt_node: nil,
    val: :main
  },
  node_name: :program,
  rt_node: nil,
  val: nil
}
```

(-t) We show token's list from source code. Must check a relational couple to recognize every token.

A terminal window with a dark background and light-colored text. The window title bar shows the user 'parisollie' and the path '~/Descargas/GoldCompilers/GoldCompilers/Co...'. The prompt is 'parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/CompiladorFoxy\$'. The command './foxy -t codec_c/StageTwo/Valid/13bitwise.c' has been executed. The output is 'The token list is :' followed by a list of tokens in a JSON-like format. The tokens are: {type, 1, [:intKeyword]}, {ident, 1, [:mainKeyword]}, {left_paren, 1, []}, {right_paren, 1, []}, {left_brace, 1, []}, {ident, 2, [:returnKeyword]}, {op, 2, [:log_Neg]}, {num, 2, 17}, {semicolon, 2, []}, and {right_brace, 3, []}.

```
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/Co...
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/CompiladorFoxy$ ./foxy -t codec_c/StageTwo/Valid/13bitwise.c

The token list is :

[
  {type, 1, [:intKeyword]},
  {ident, 1, [:mainKeyword]},
  {left_paren, 1, []},
  {right_paren, 1, []},
  {left_brace, 1, []},
  {ident, 2, [:returnKeyword]},
  {op, 2, [:log_Neg]},
  {num, 2, 17},
  {semicolon, 2, []},
  {right_brace, 3, []}
]
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/CompiladorFoxy$
```

(-s)Assembly must write in 64-bits set instructions.
The assembly syntax must be a AT and T by default in GCC.

```
parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/Co...
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/Compil
adorFoxy$ ./foxy -s codec_c/StageTwo/Valid/13bitwise.c

The assembly code is :

.section      __TEXT,__text,regular,nat_Instructs
.p2align     4, 0x90
.globl _main      ## Begin function main
_main:      ## @main
movl    17, %eax
push    %rax
pop     %rbx
cmpl    $0, %eax
movl    $0, %eax
sete    %al
push    %rax
pop     %rbx
ret
push    %rax
pop     %rbx
push    %rax
pop     %rbx

parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/Compil
```

(-c) We get the assmbler and the executable:

```
parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/GoldCompilers/C...
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/GoldCompilers/C...
ompiladorFoxy$ ./foxy -c codec_c/StageTwo/Valid/13bitwise.c

*****

The compiling the file is :
codec_c/StageTwo/Valid/13bitwise.c

*****

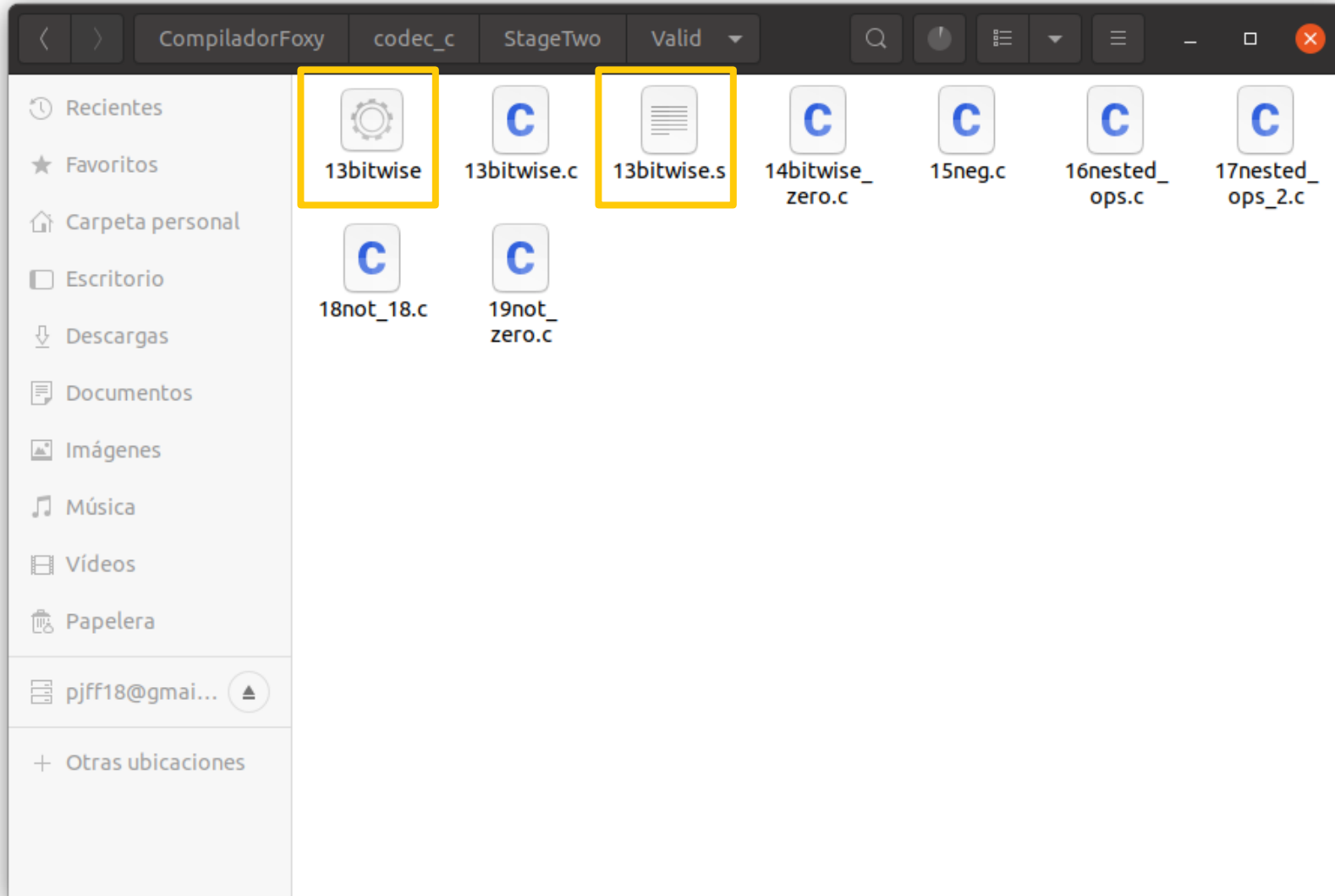
The program has worked correctly :D, your executables have been generated:
.....

The asambler it's located in the following route: codec_c/StageTwo/Valid/13bitwise.s
.....

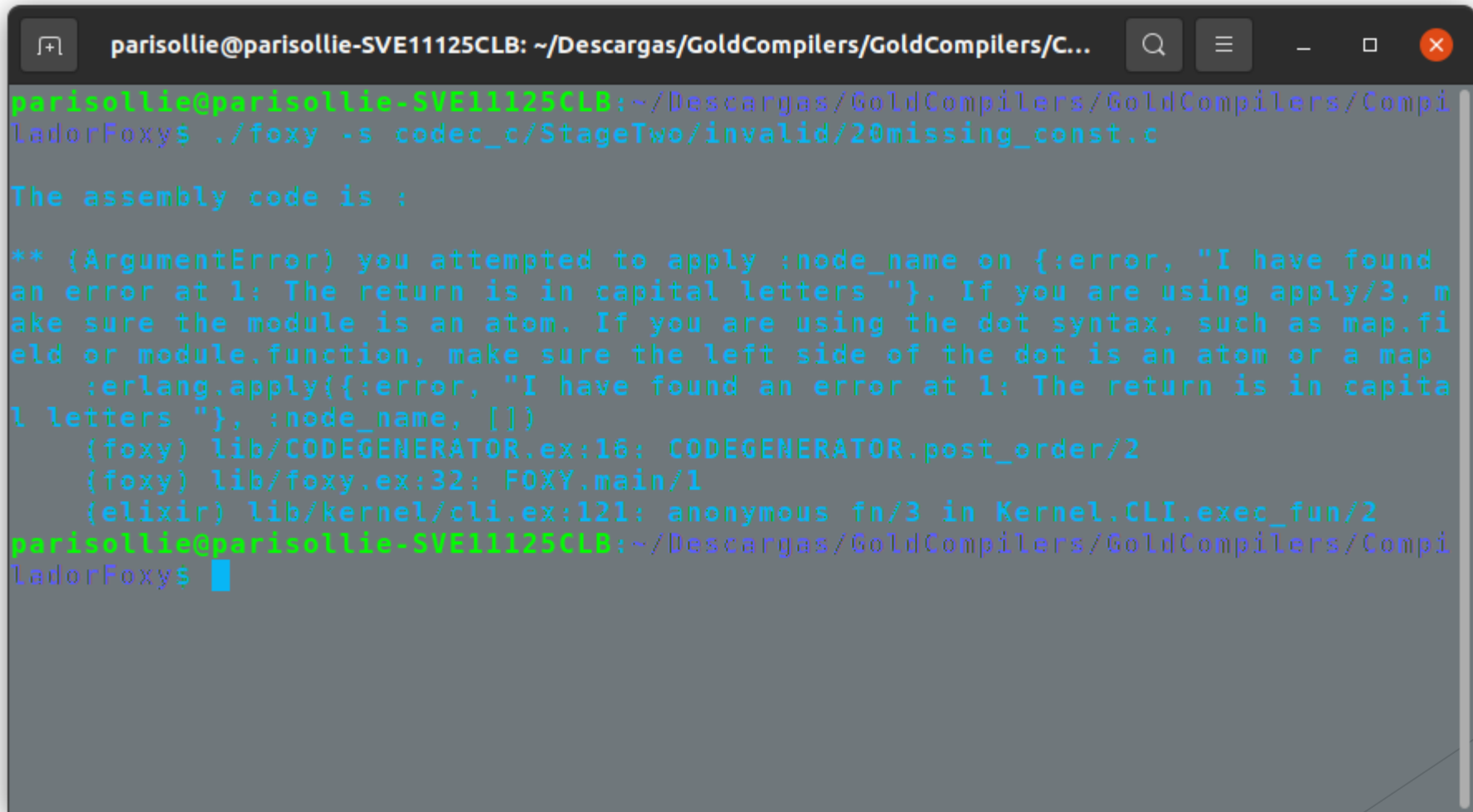
The executable it's located in the following route: codec_c/StageTwo/Valid/./13bitwise
.....

parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/GoldCompilers/C...
ompiladorFoxy$
```

It is generated in our folder:



Test Invalid:



```
parisollie@parisollie-SVE11125CLB: ~/Descargas/GoldCompilers/GoldCompilers/C...
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/Compi
LadorFoxy$ ./foxy -s codec_c/StageTwo/invalid/20missing_const.c

The assembly code is :

** (ArgumentError) you attempted to apply :node_name on {:error, "I have found
an error at 1: The return is in capital letters "}. If you are using apply/3, m
ake sure the module is an atom. If you are using the dot syntax, such as map.fi
eld or module.function, make sure the left side of the dot is an atom or a map
:erlang.apply({:error, "I have found an error at 1: The return is in capita
l letters "}, :node_name, [])
    (foxy) lib/CODEGENERATOR.ex:16: CODEGENERATOR.post_order/2
    (foxy) lib/foxy.ex:32: FOXY.main/1
    (elixir) lib/kernel/cli.ex:121: anonymous fn/3 in Kernel.CLI.exec_fun/2
parisollie@parisollie-SVE11125CLB:~/Descargas/GoldCompilers/GoldCompilers/Compi
LadorFoxy$
```




► Thank you !