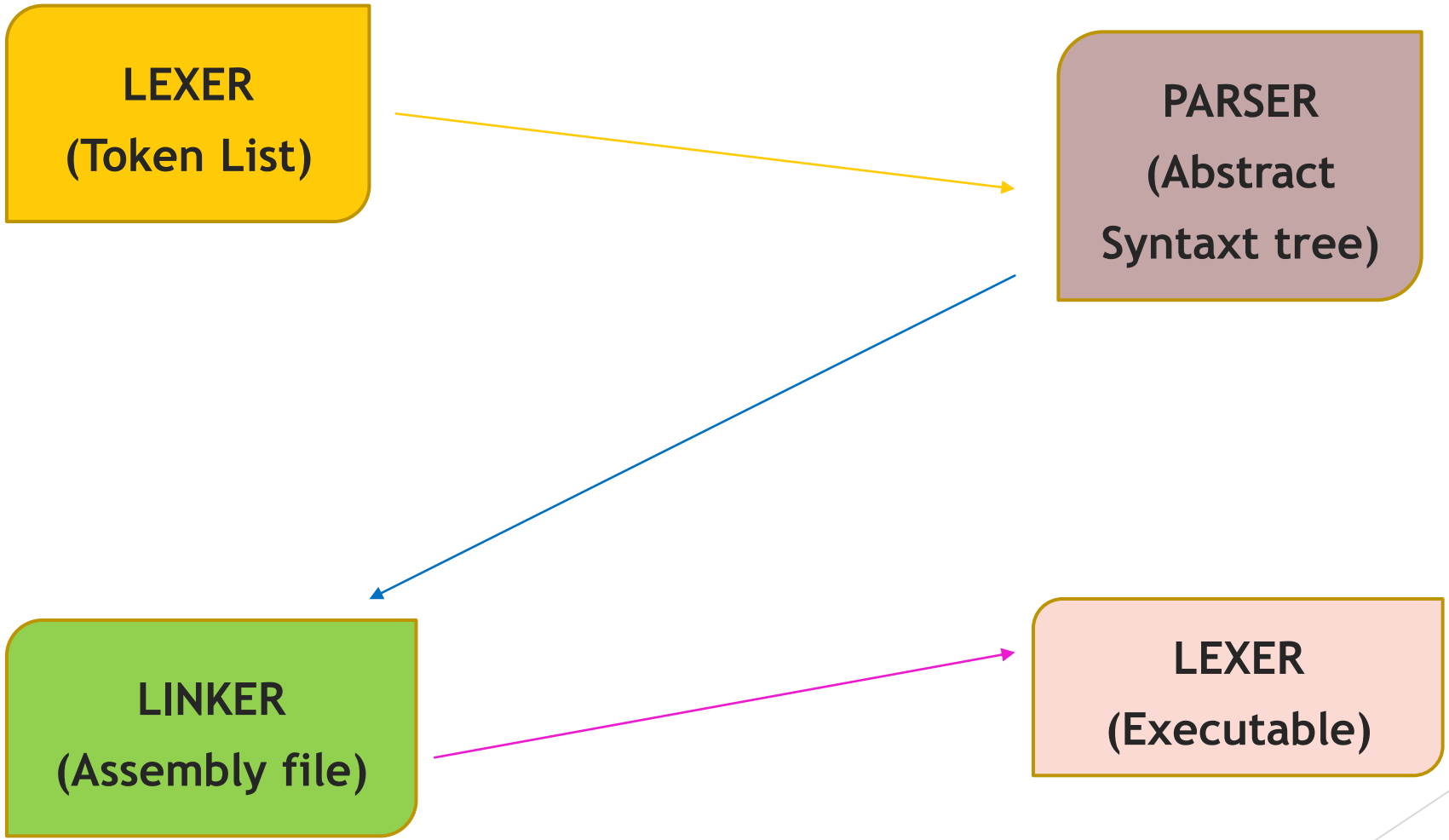
Gold Compilers: Compiler “Foxy”

All the stages

Integrants:

- García Felipe Miguel (Project Manager)
- Felix Flores Paul Jaime (Tester)
- San Juan 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

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:

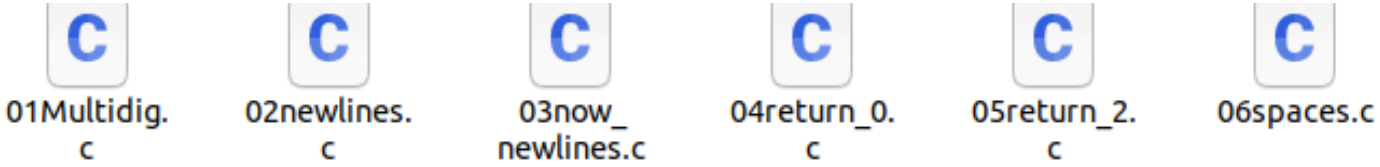
```
{:type, :intKeyWord},#                  int
{:ident, :returnKeyWord},#              return
{:ident, :mainKeyWord},#                main
{:left_brace},#                         {
{:right_brace},#                        }
{:left_paren},#                         (
{:right_paren},#                        )
{:semicolon},#                          ;
############################ Segunda entrega ##
```

Test that the compiler must to do:

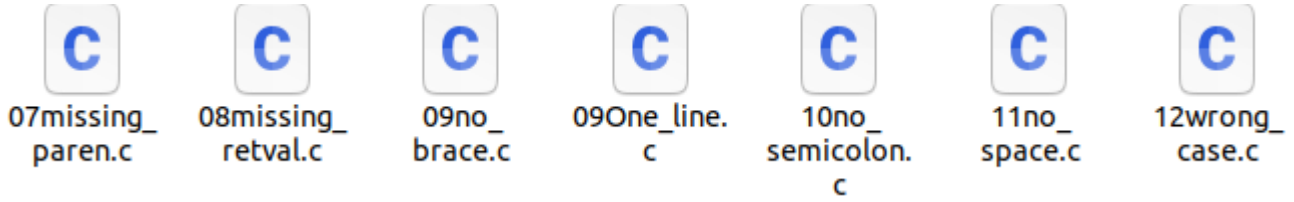
- You can compile several tests that we have, if you want it in the same way, but here we will only do one

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

Valids :



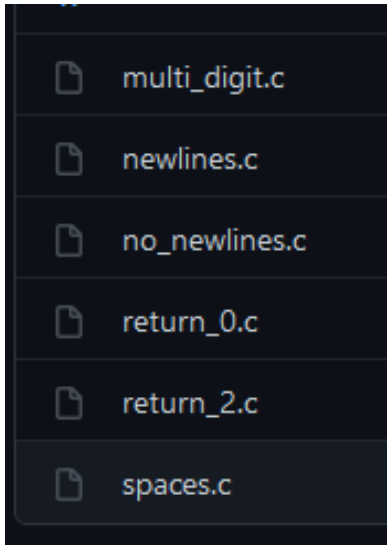
Invalids:



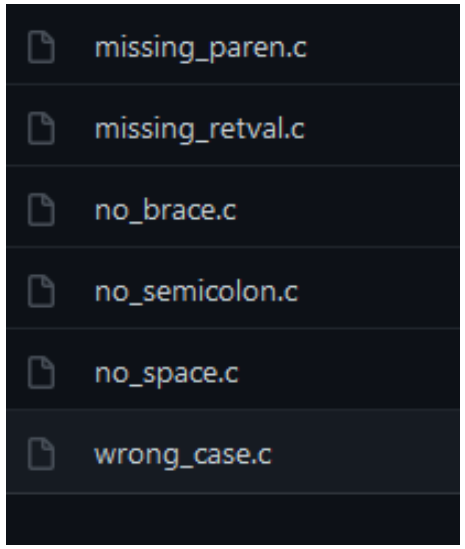
Test of Nora first stage:

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

- Valids:



- Invalids:



Example stage one:

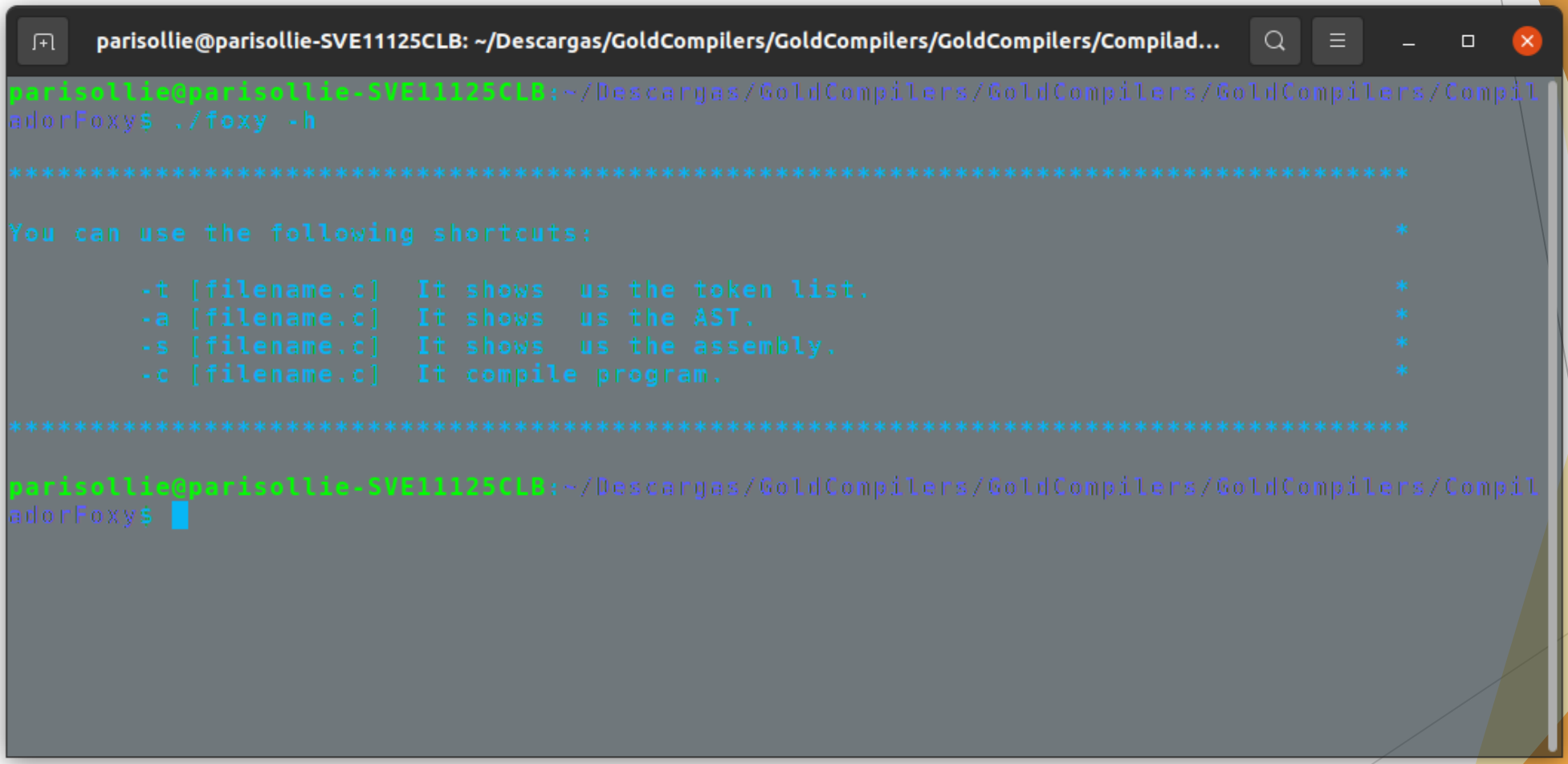
Valid example:

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

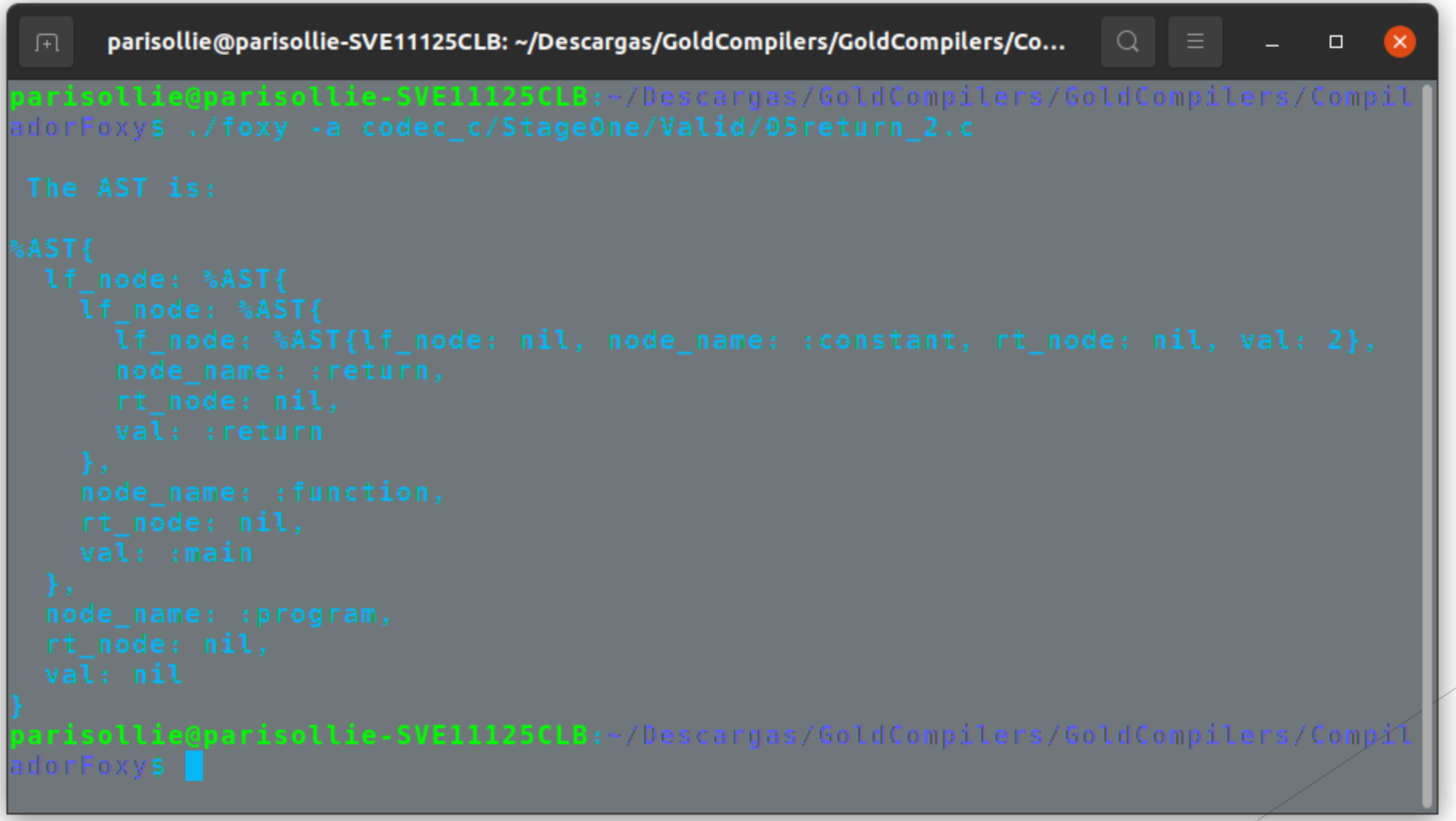
Invalid example:

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

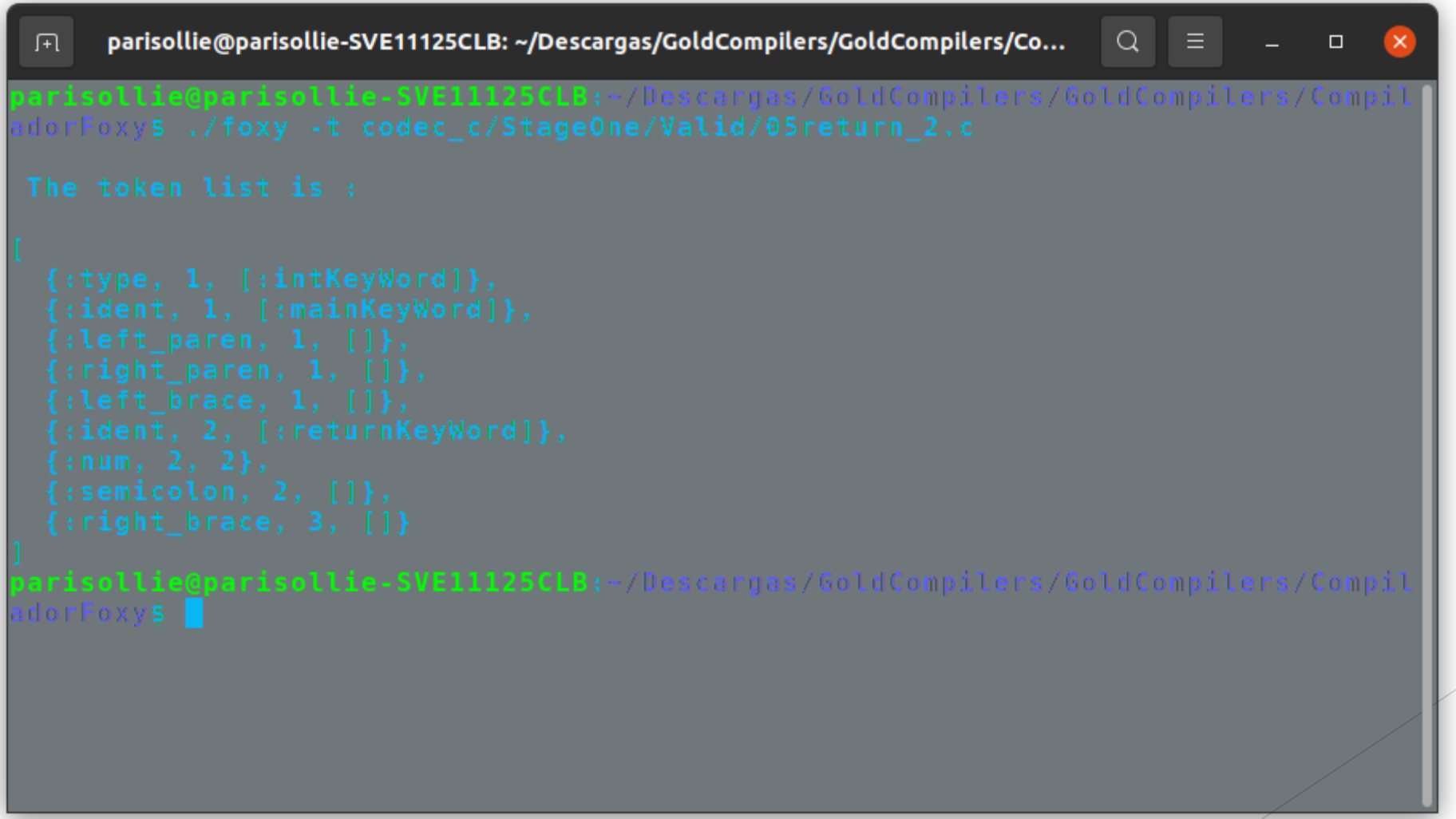
How to use the compiler Foxy with:



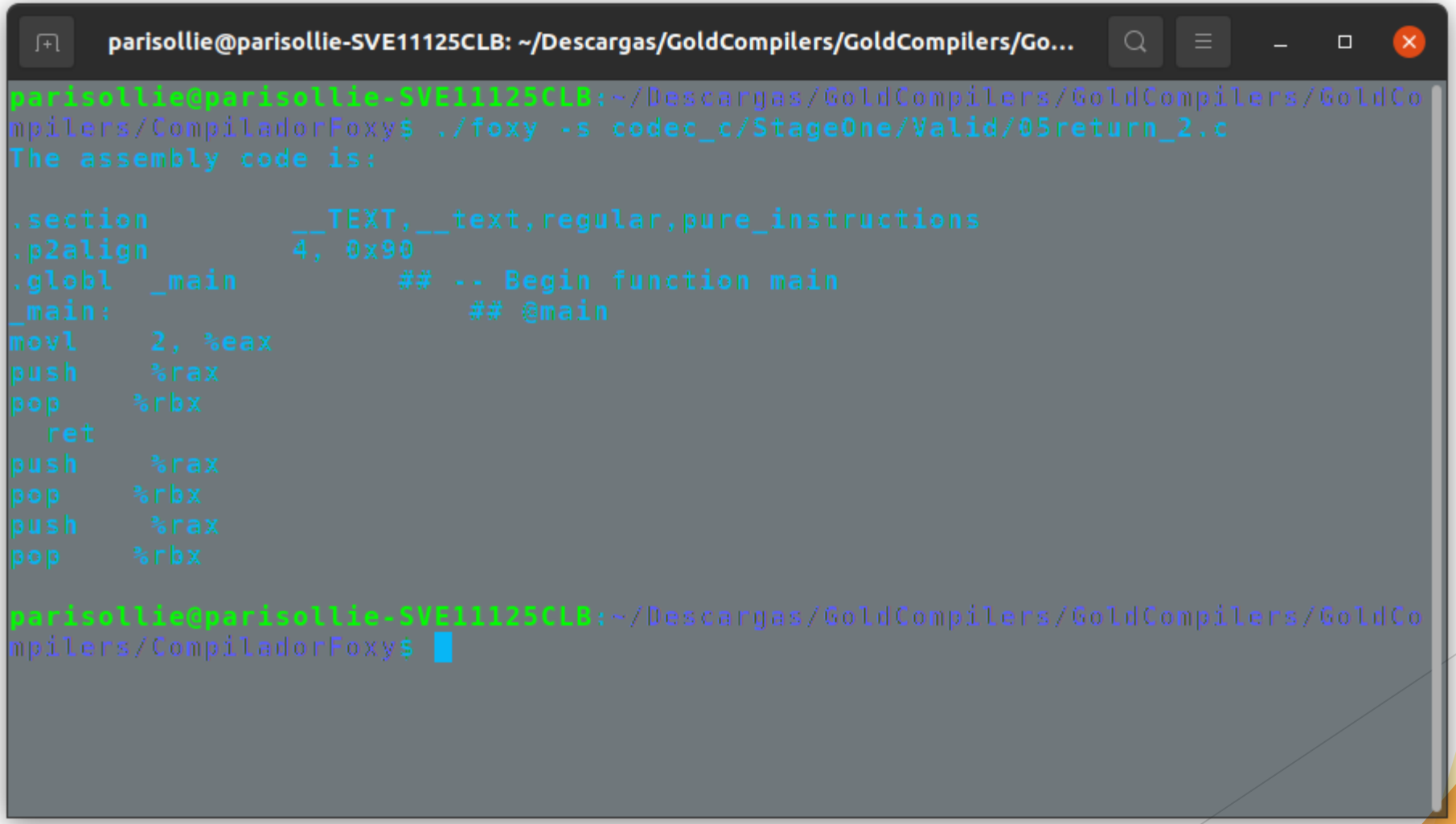
(-a)Development language must be a matching pattern to easily build an Abstract Syntax Tree (AST), however, phase I the right side's tree must be nil.



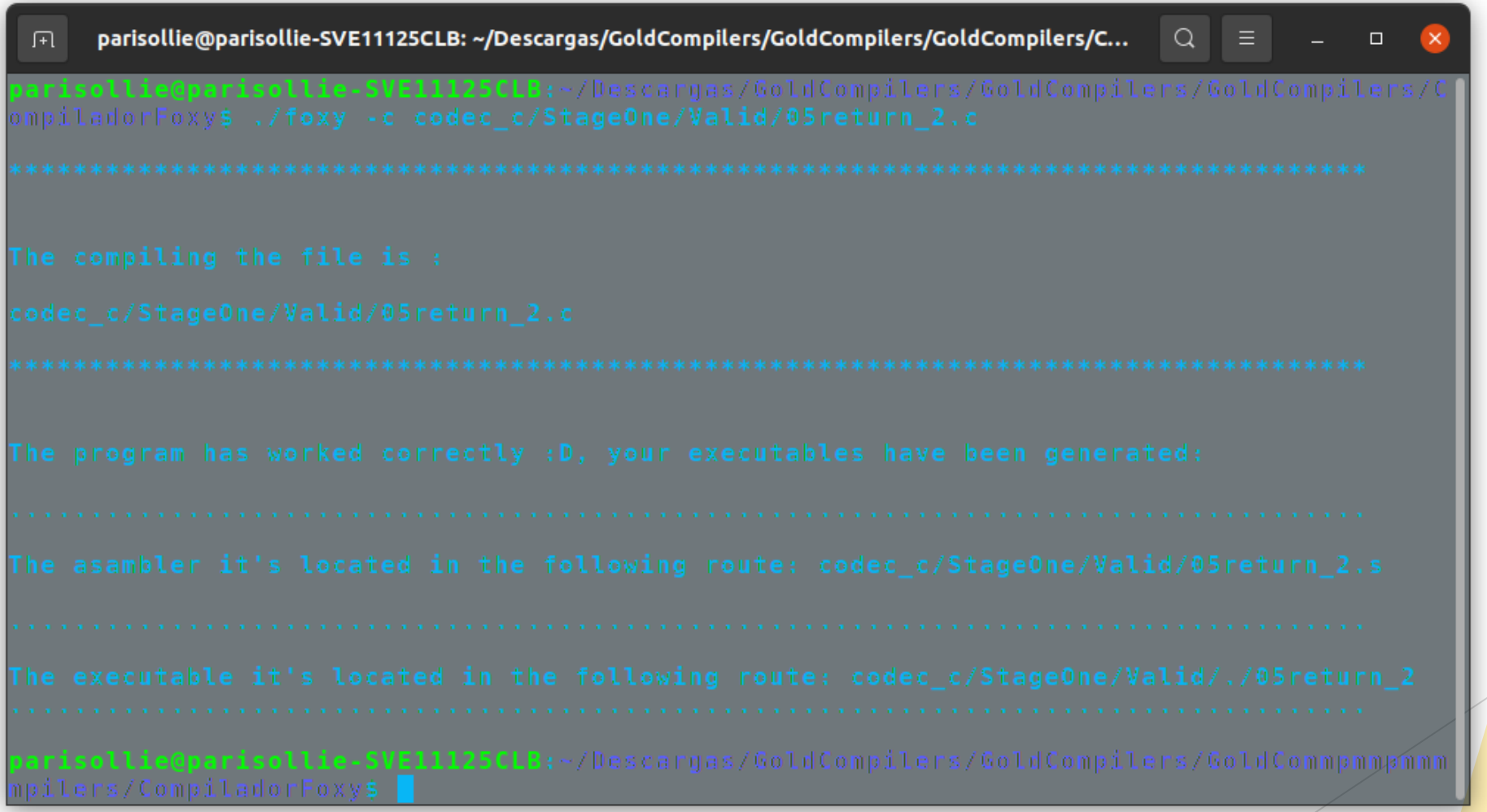
(-t)We show token's list form source code. Must check a relational couple to recognize every token.



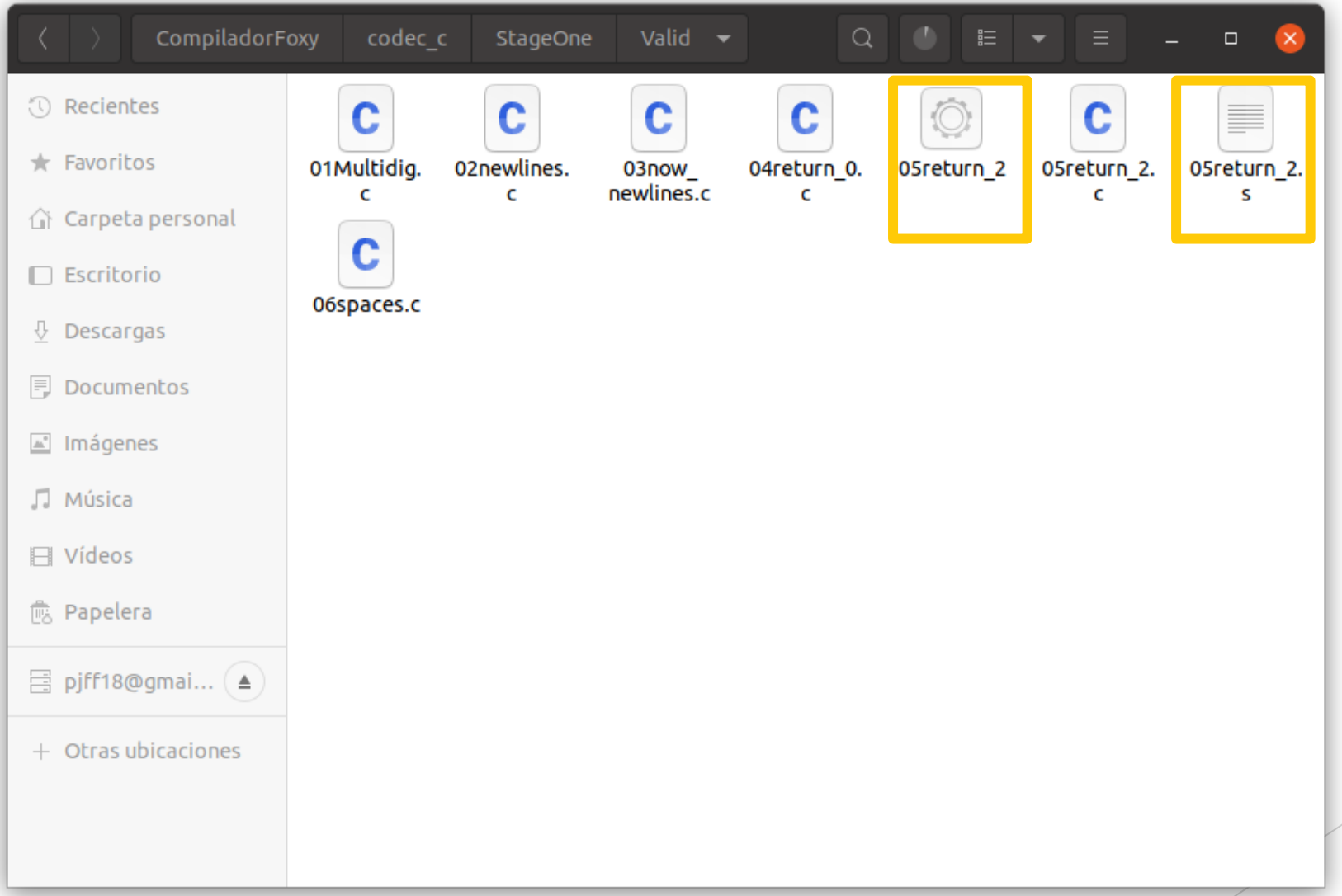
(-s)Assembly must write in 64-bits set instructions.
The assembly syntax must be a AT and T by default in GCC.



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



It is generated in our folder:

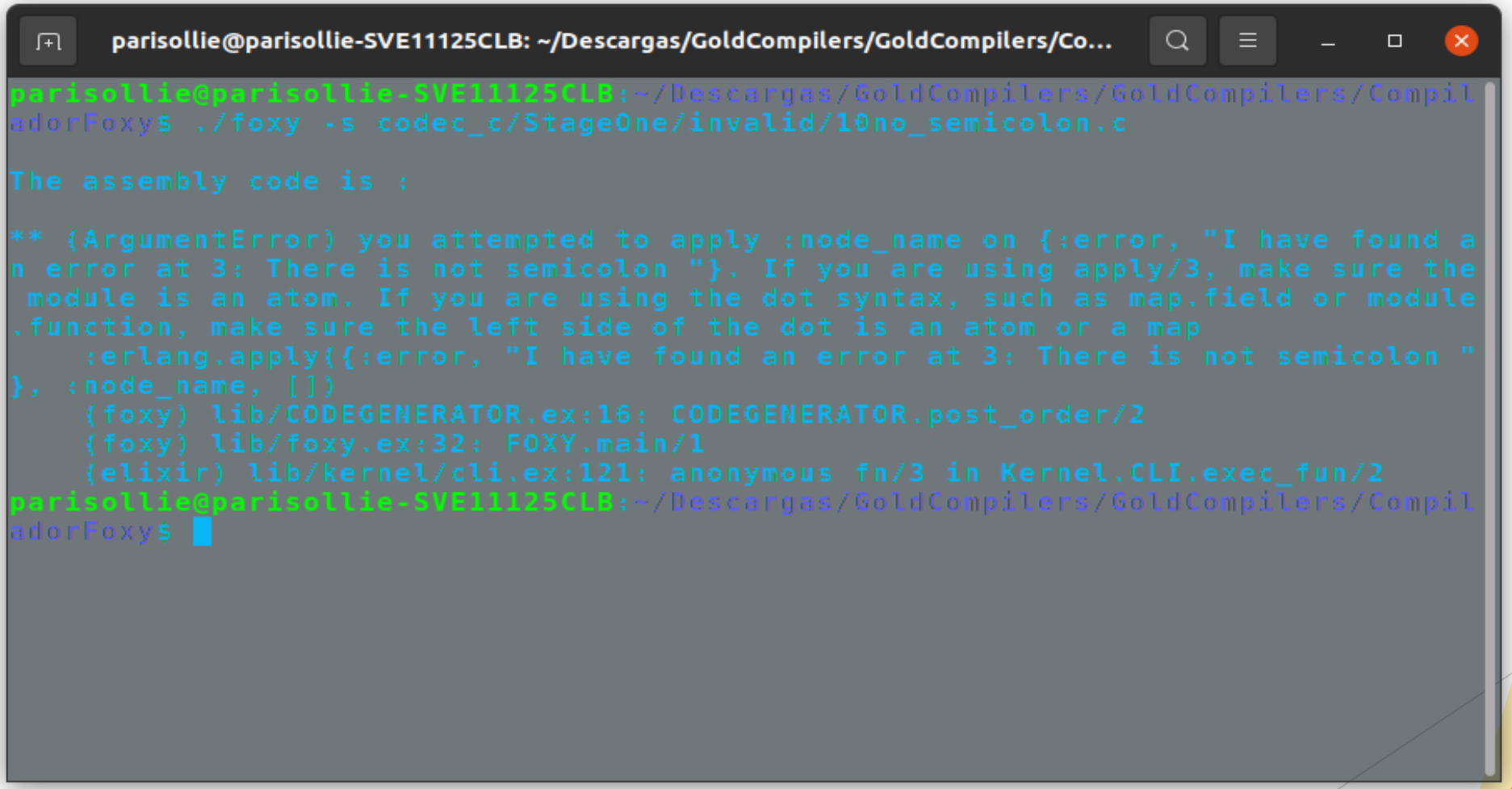


echo %errorlevel%
check the return code; it should be 2

```
C:\Users\Henry\Desktop\CompiladorFoxy\codec_c\StageOne\Valid>05return_2.exe

C:\Users\Henry\Desktop\CompiladorFoxy\codec_c\StageOne\Valid>echo %errorlevel%
2
```

Test Invalid:



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 ########
```

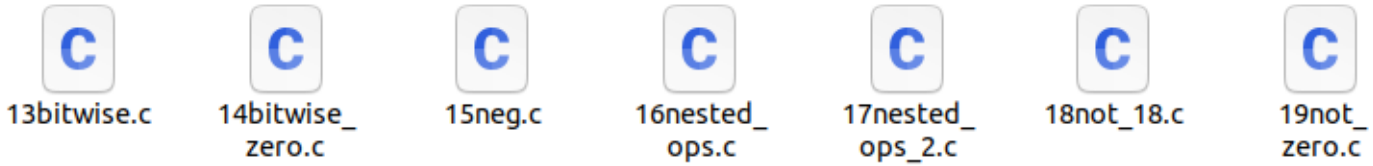
Test that the compiler must to do:

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

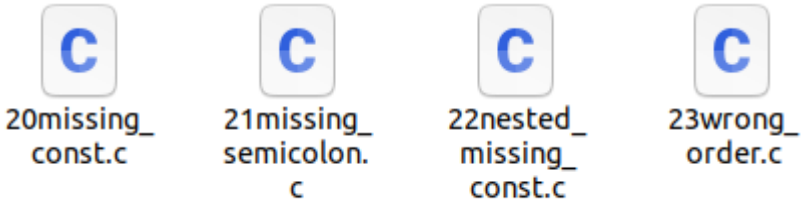
  You can see the examples valid e invalids in the

  folders:

Valids :



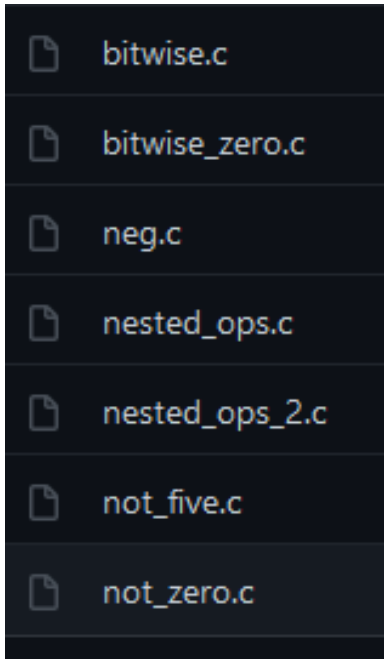
Invalids:



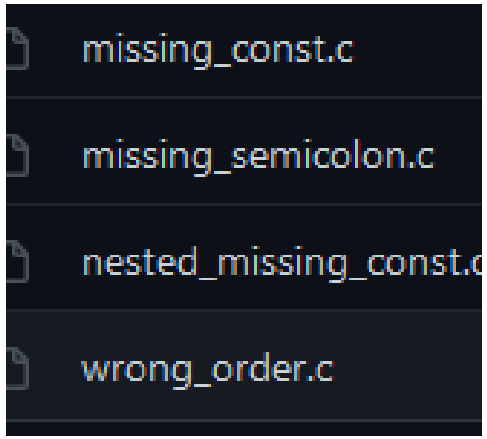
Test of Nora second stage:

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

- Valids:



- Invalids:



Example stage two:

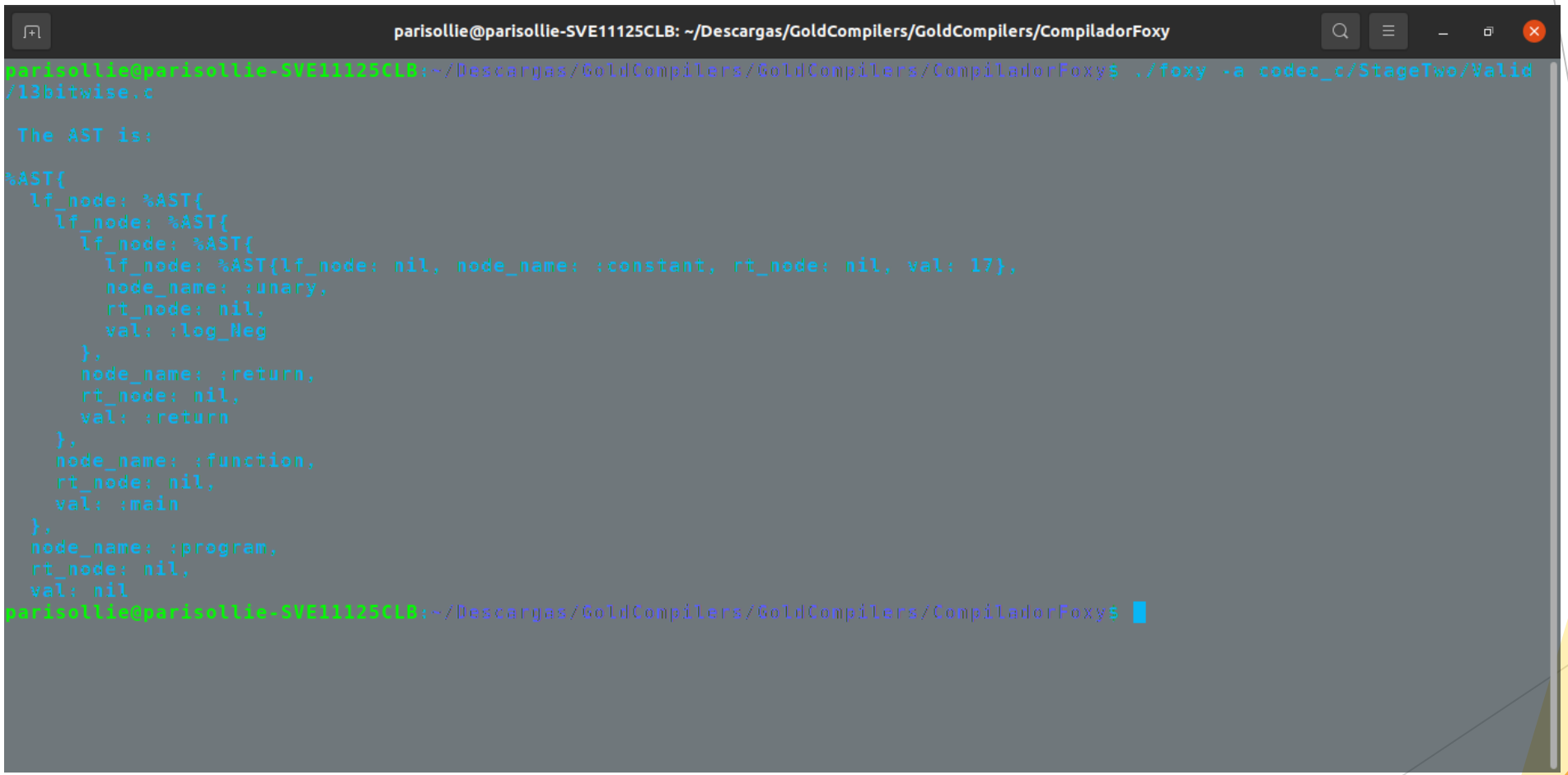
Valid example:

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

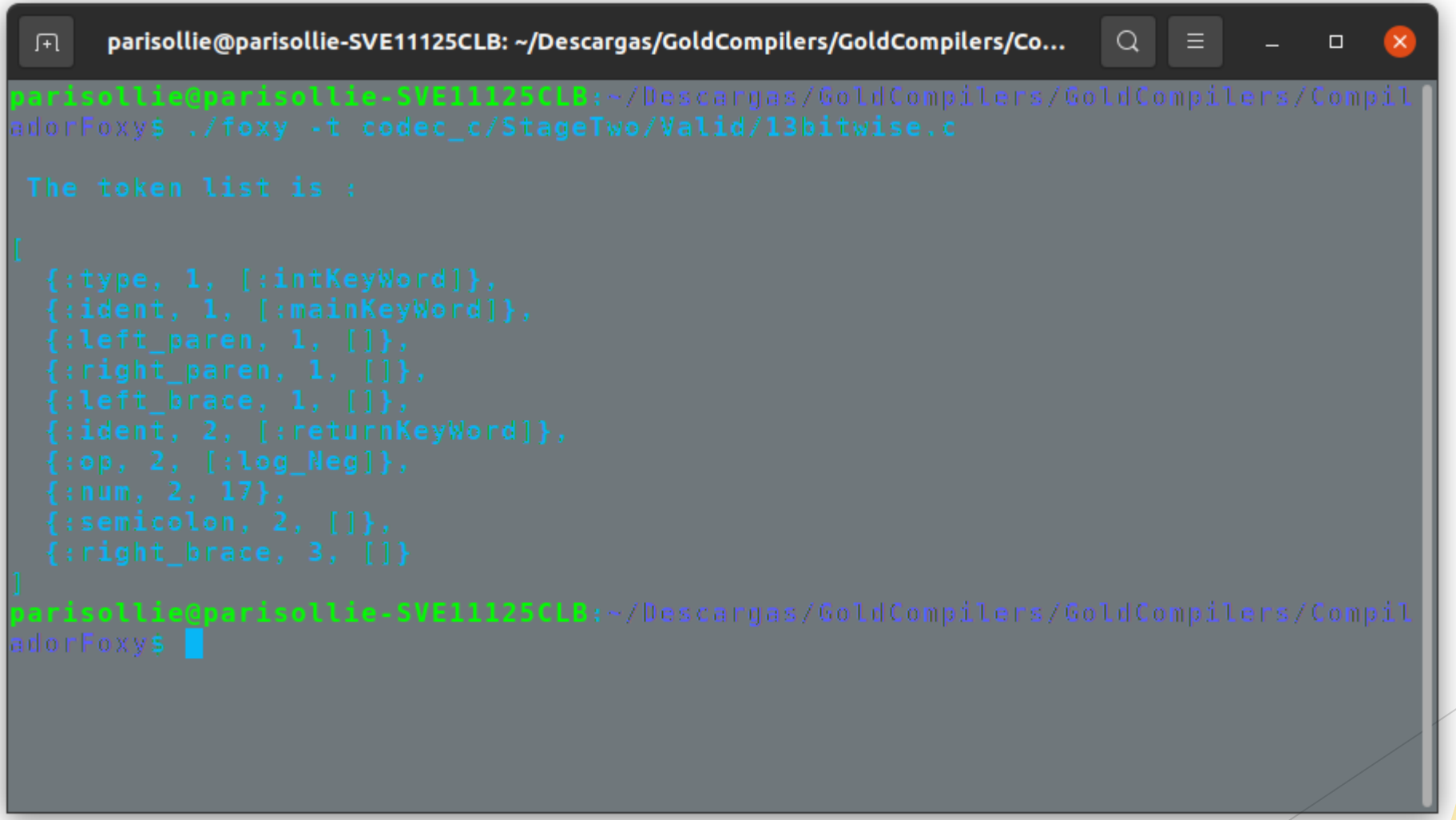
Invalid example:

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

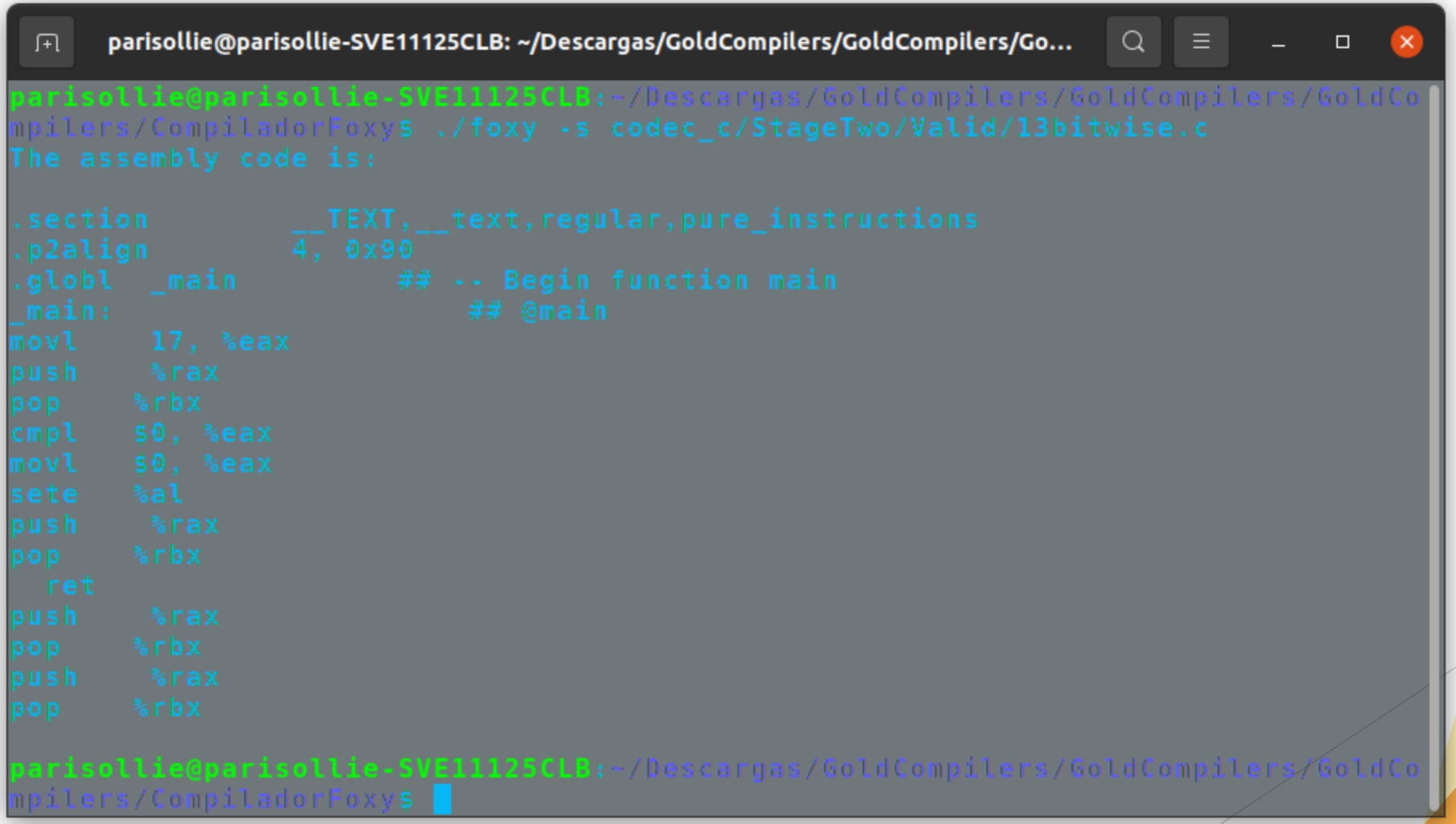
(-a)Development language must be a matching pattern to easily build an Abstract Syntax Tree (AST), however, phase I the right side's tree must be nil.



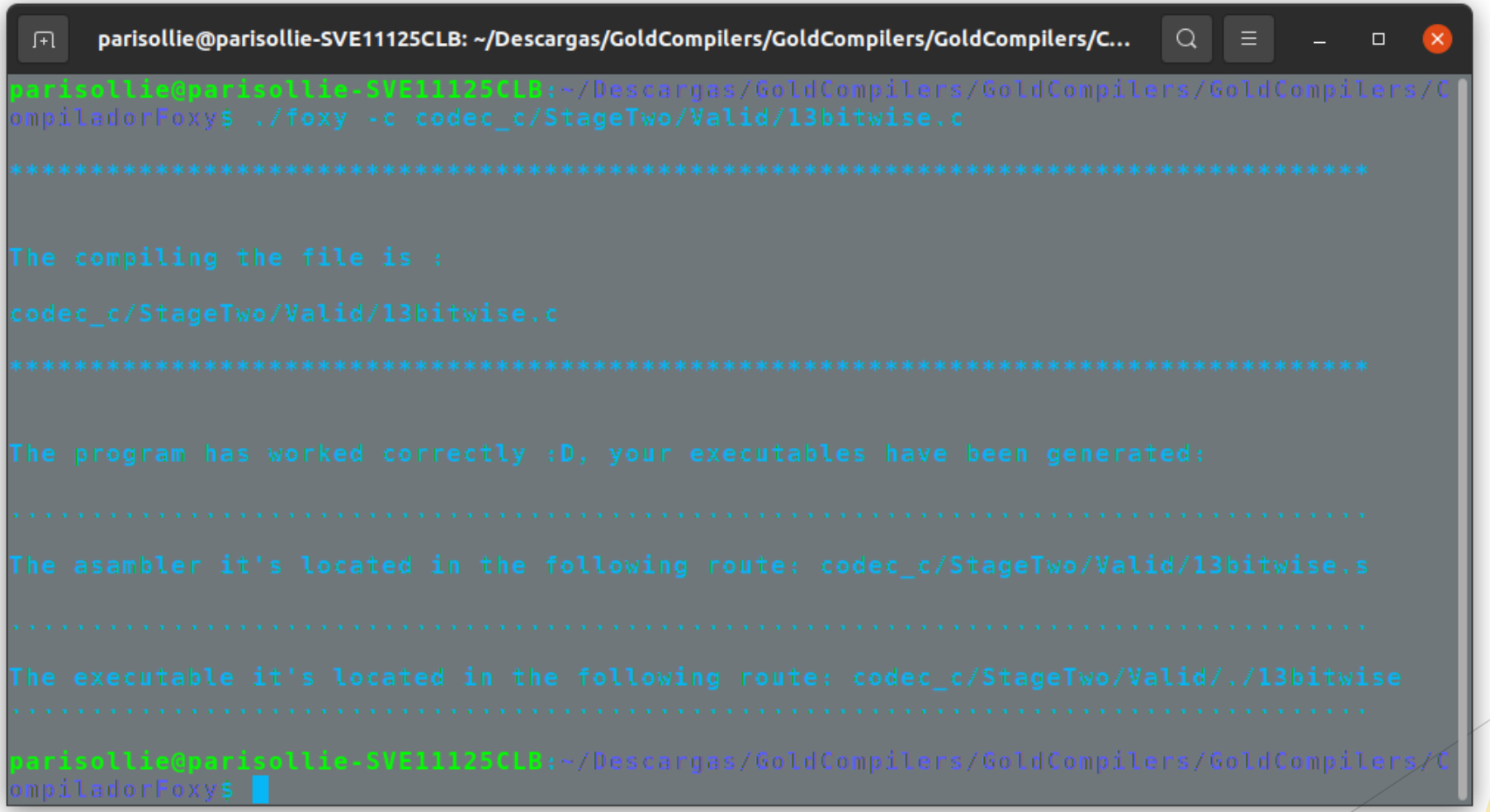
(-t)We show token's list form source code. Must check a relational couple to recognize every token.



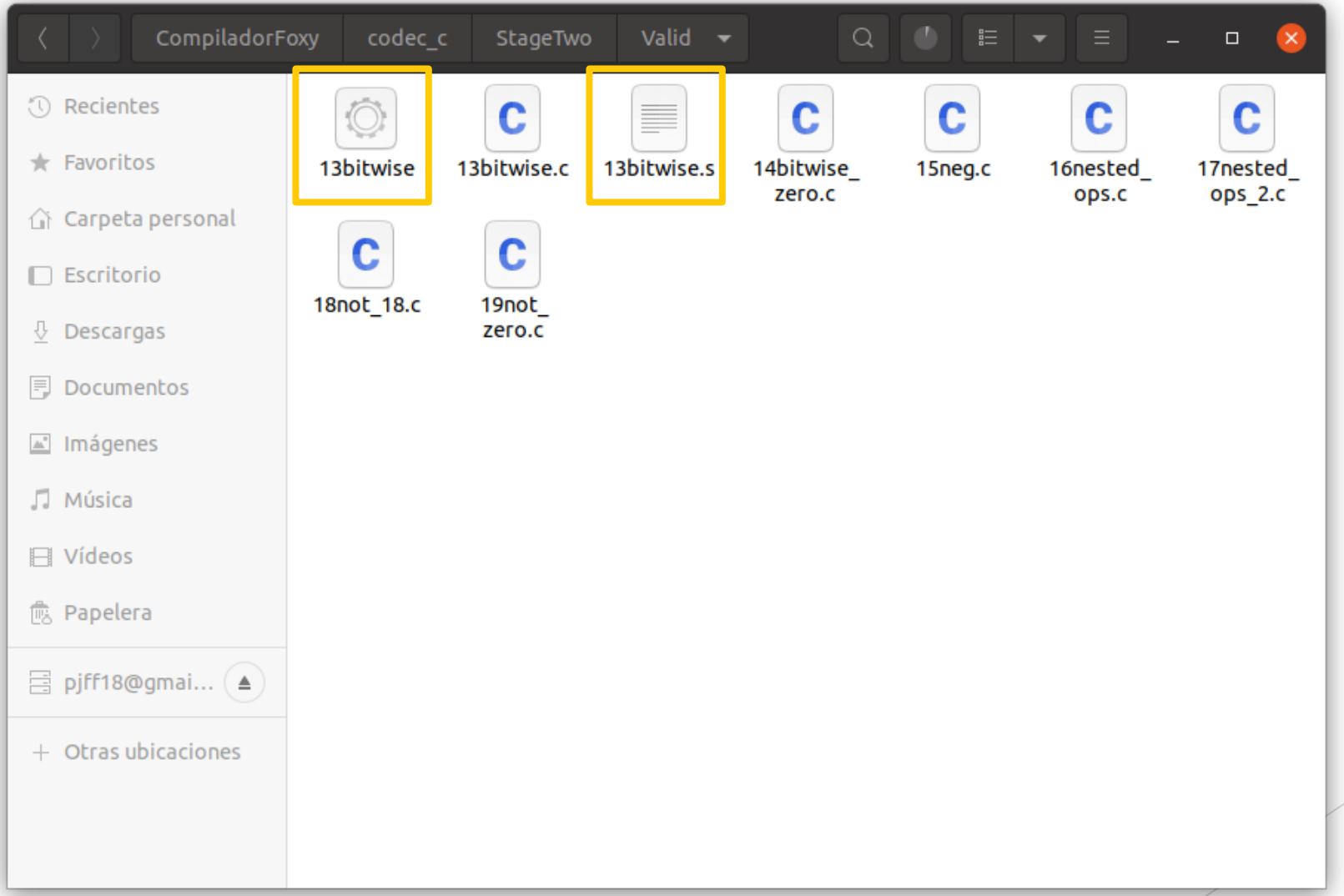
(-s)Assembly must write in 64-bits set instructions.
The assembly syntax must be a AT and T by default in GCC.



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



It is generated in our folder:

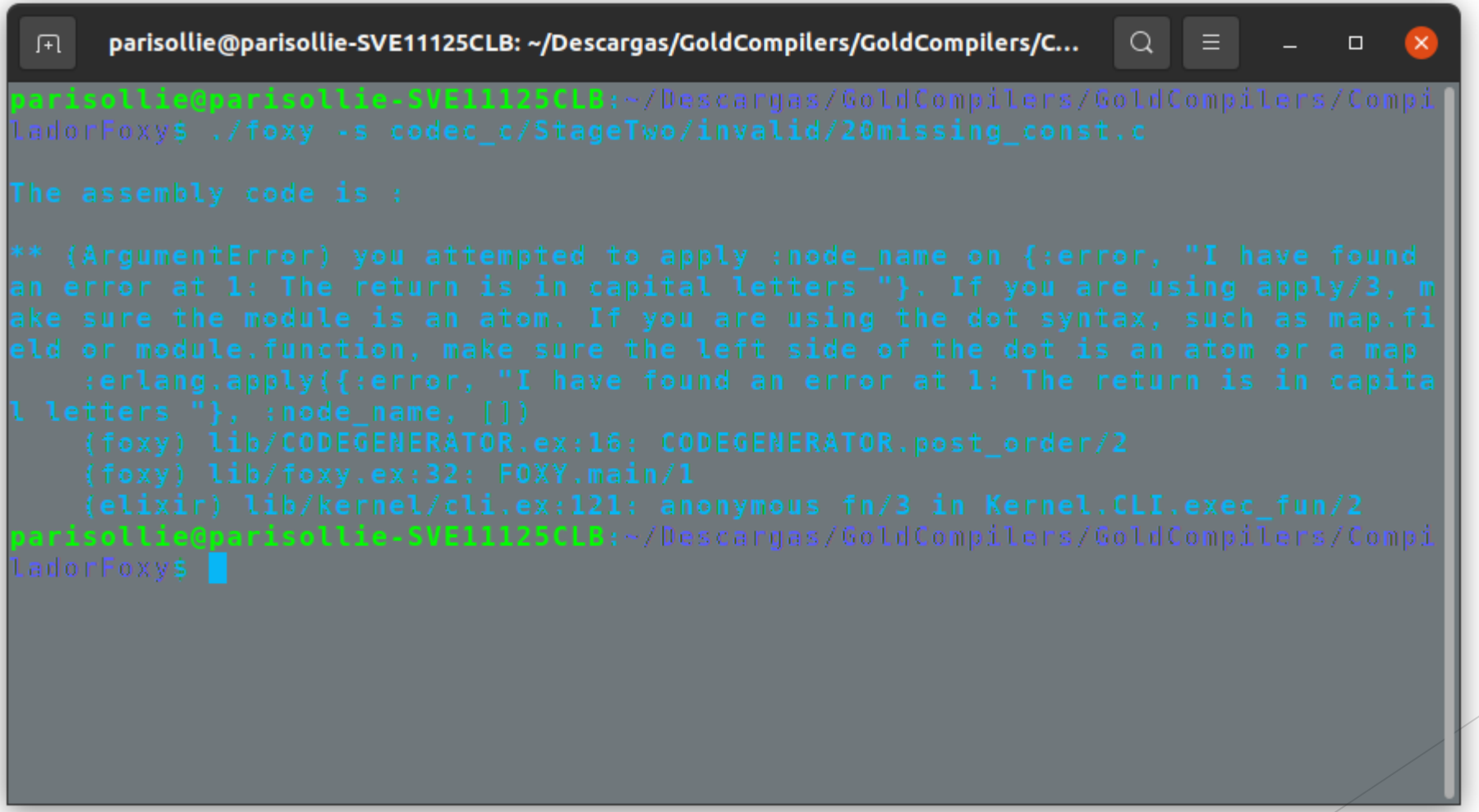


echo %errorlevel%
check the return code; it should be 0

```
C:\Users\Henry\Desktop\CompiladorFoxy\codec_c\StageTwo\Valid>13bitwise.exe

C:\Users\Henry\Desktop\CompiladorFoxy\codec_c\StageTwo\Valid>echo %errorlevel%
0
```

Test Invalid:



Stage three:

Third 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.

Third Implentation:

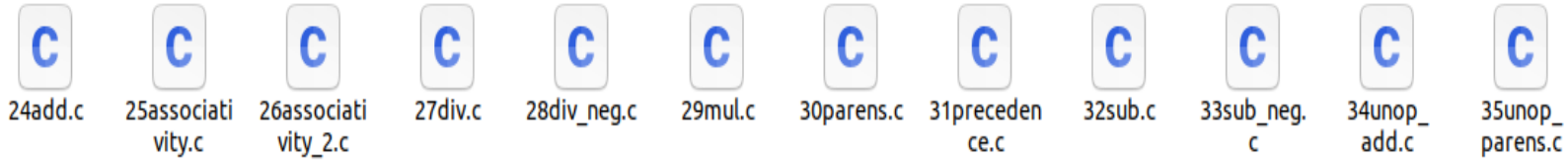
```
  ################################## Tercera entrega #
  {:op, :add},#                                  +
  {:op, :div},#                                  /
  {:op, :mul},#                                  *
]
```

Test that the compiler must to do:

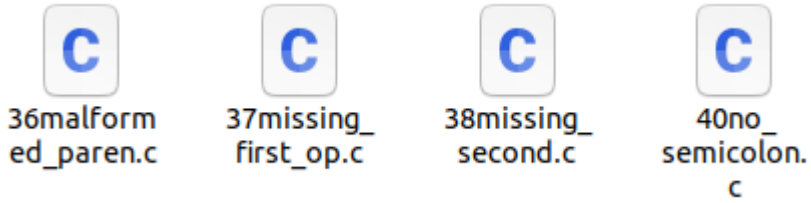
- You can compile several tests that we have, if you want it in the same way, but here we will only do one

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

Valids :



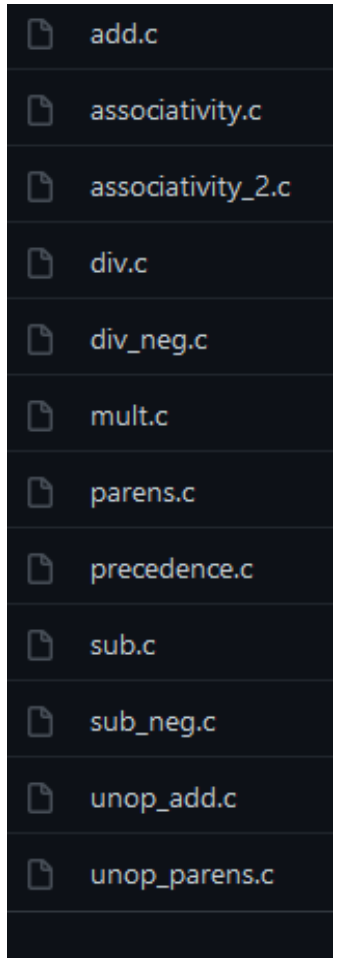
Invalids:



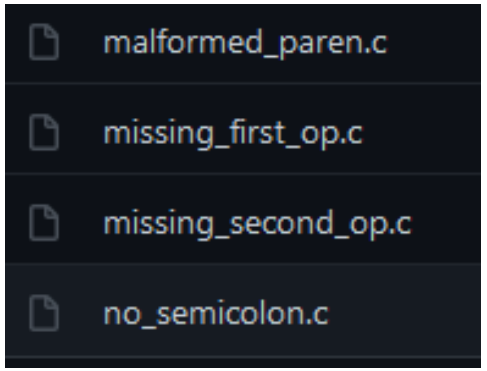
Test of Nora third stage:

https://github.com/nlsandler/write_a_c_compiler/tree/master/stage_3

- Valids:



- Invalids:



Example stage three:

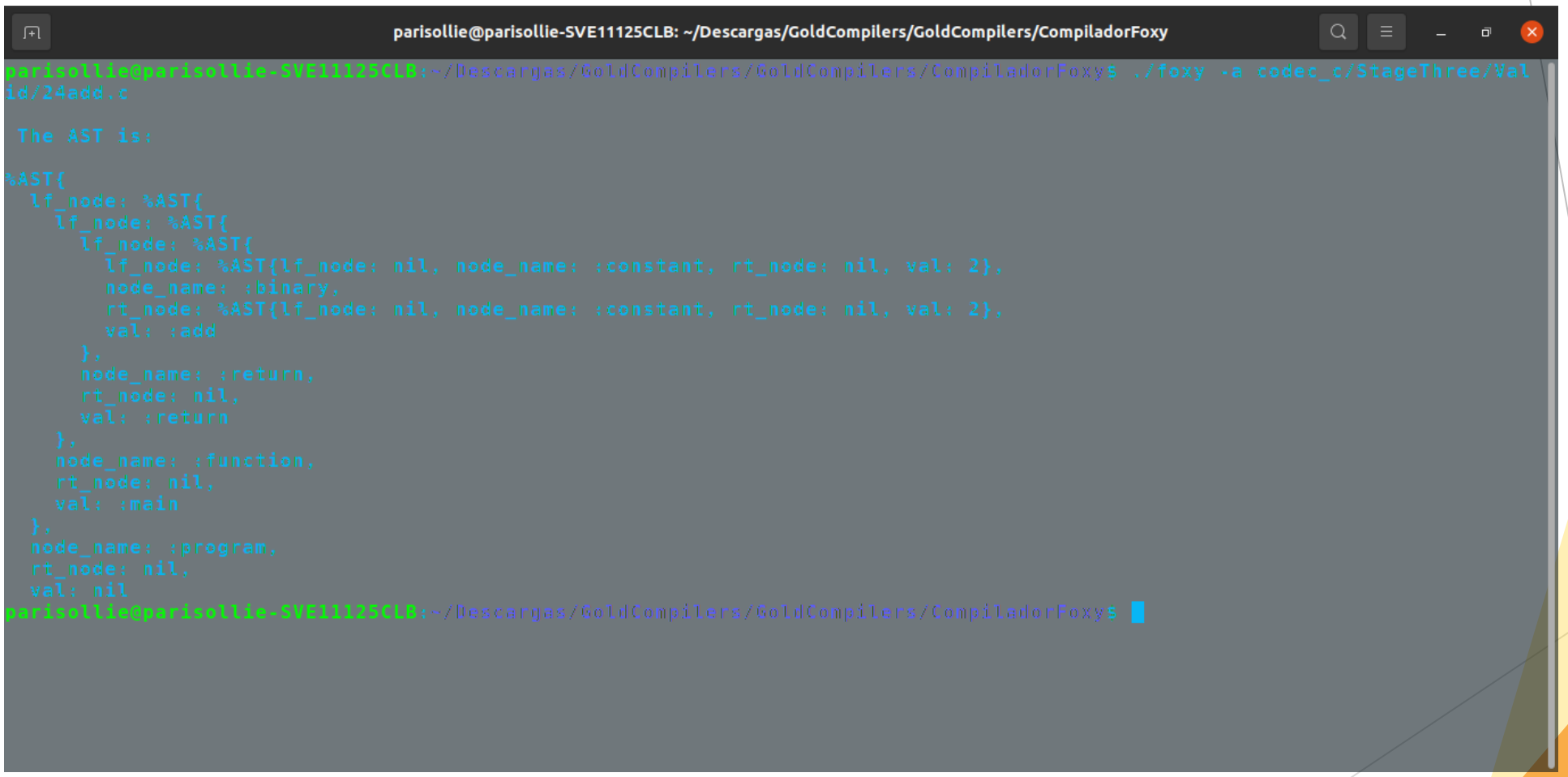
Valid example:

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

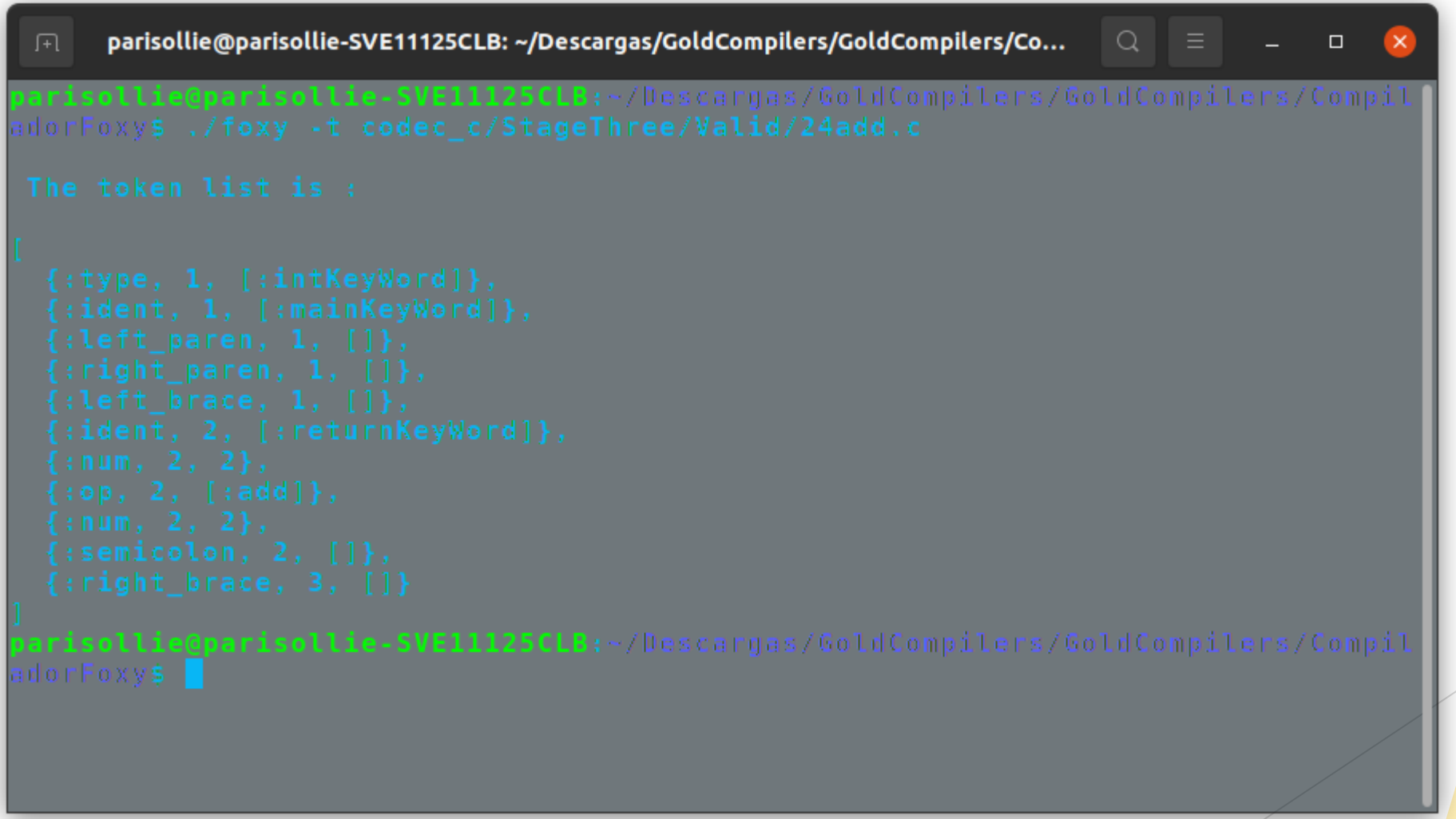
Invalid example:

```
int main( {
    return 9 (-4);
    }
```

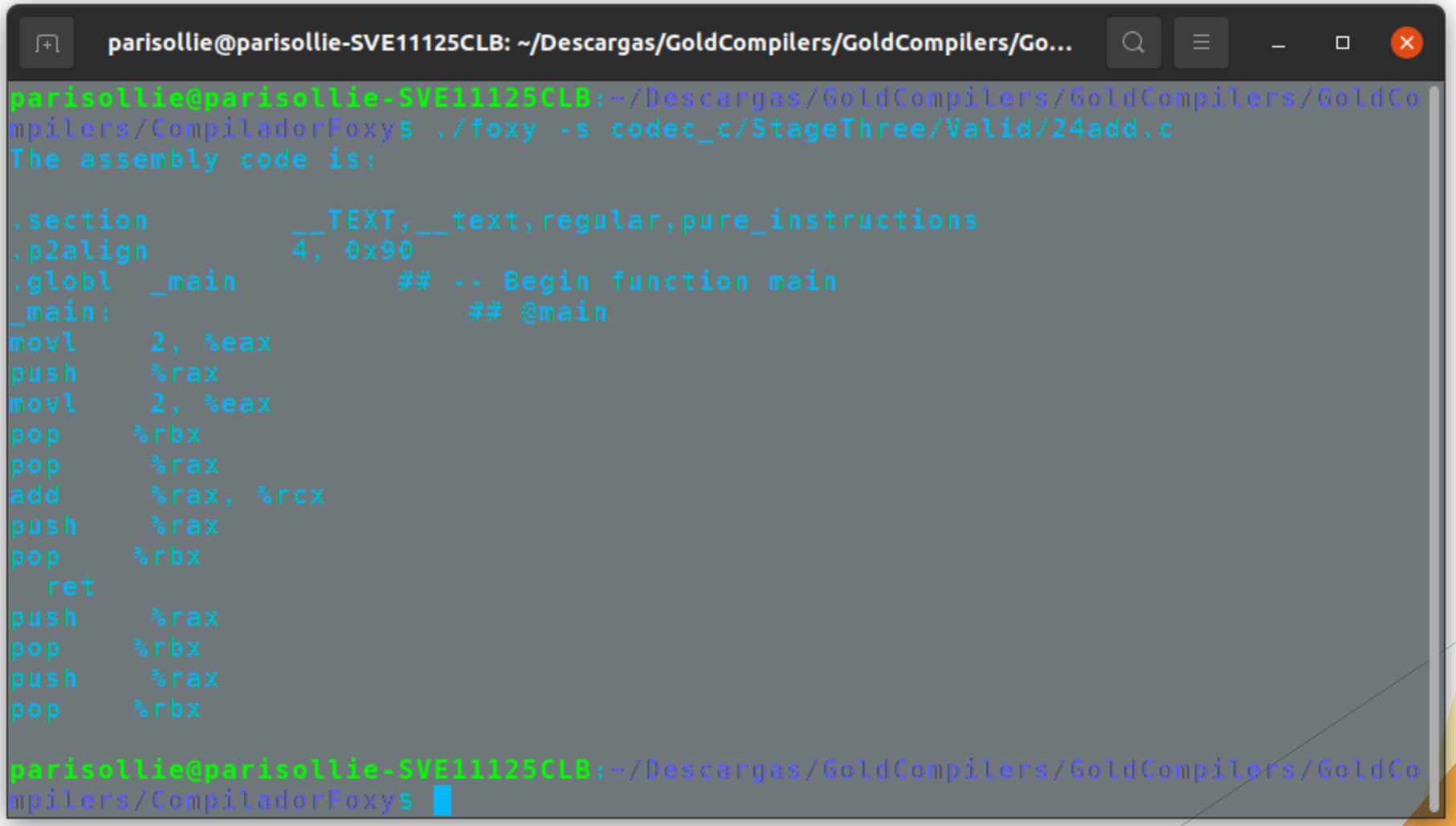
(-a)Development language must be a matching pattern to easily build an Abstract Syntax Tree (AST), however, phase I the right side's tree must be nil.



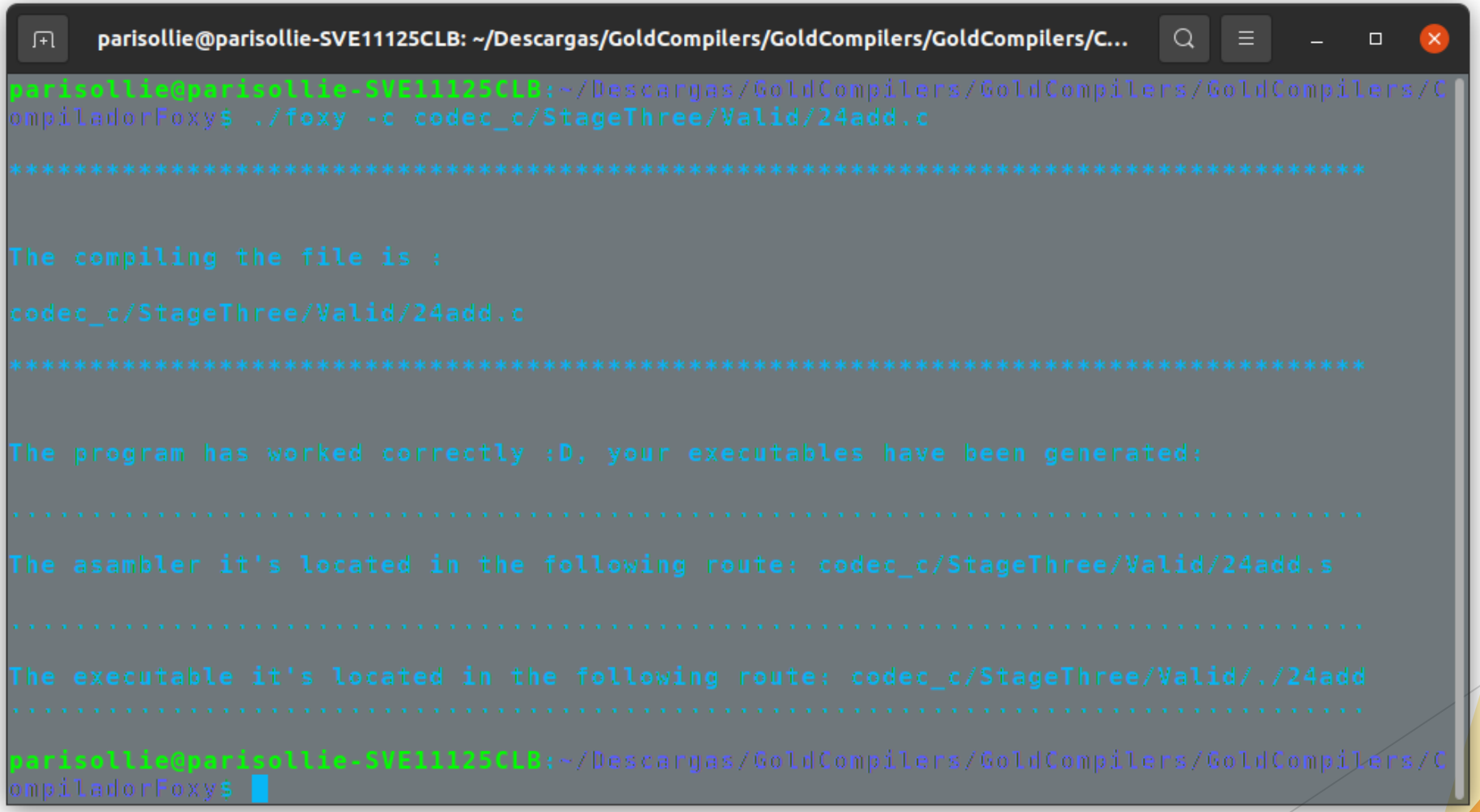
(-t)We show token's list form source code. Must check a relational couple to recognize every token.



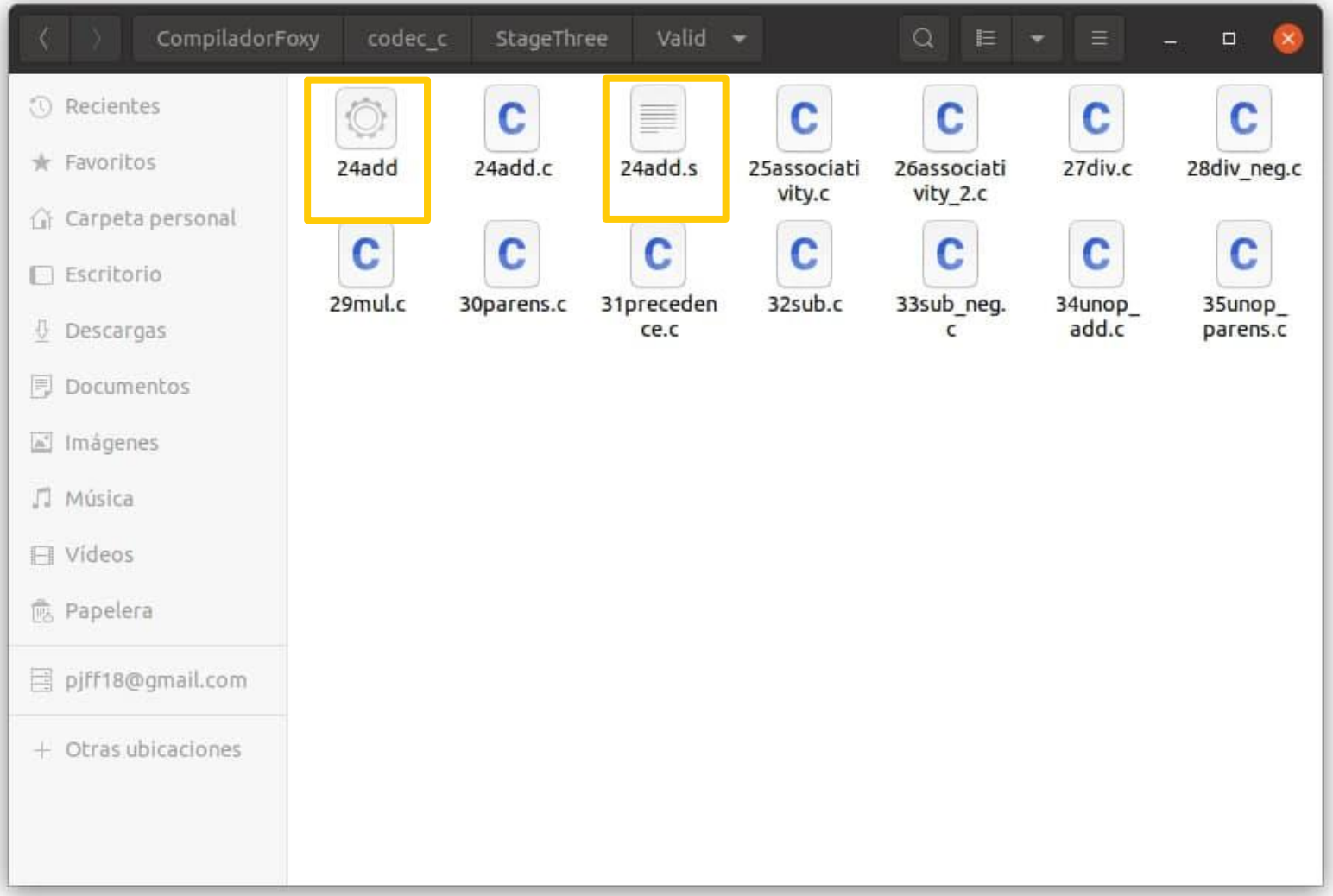
(-s)Assembly must write in 64-bits set instructions.
The assembly syntax must be a AT and T by default in GCC.



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



It is generated in our folder:

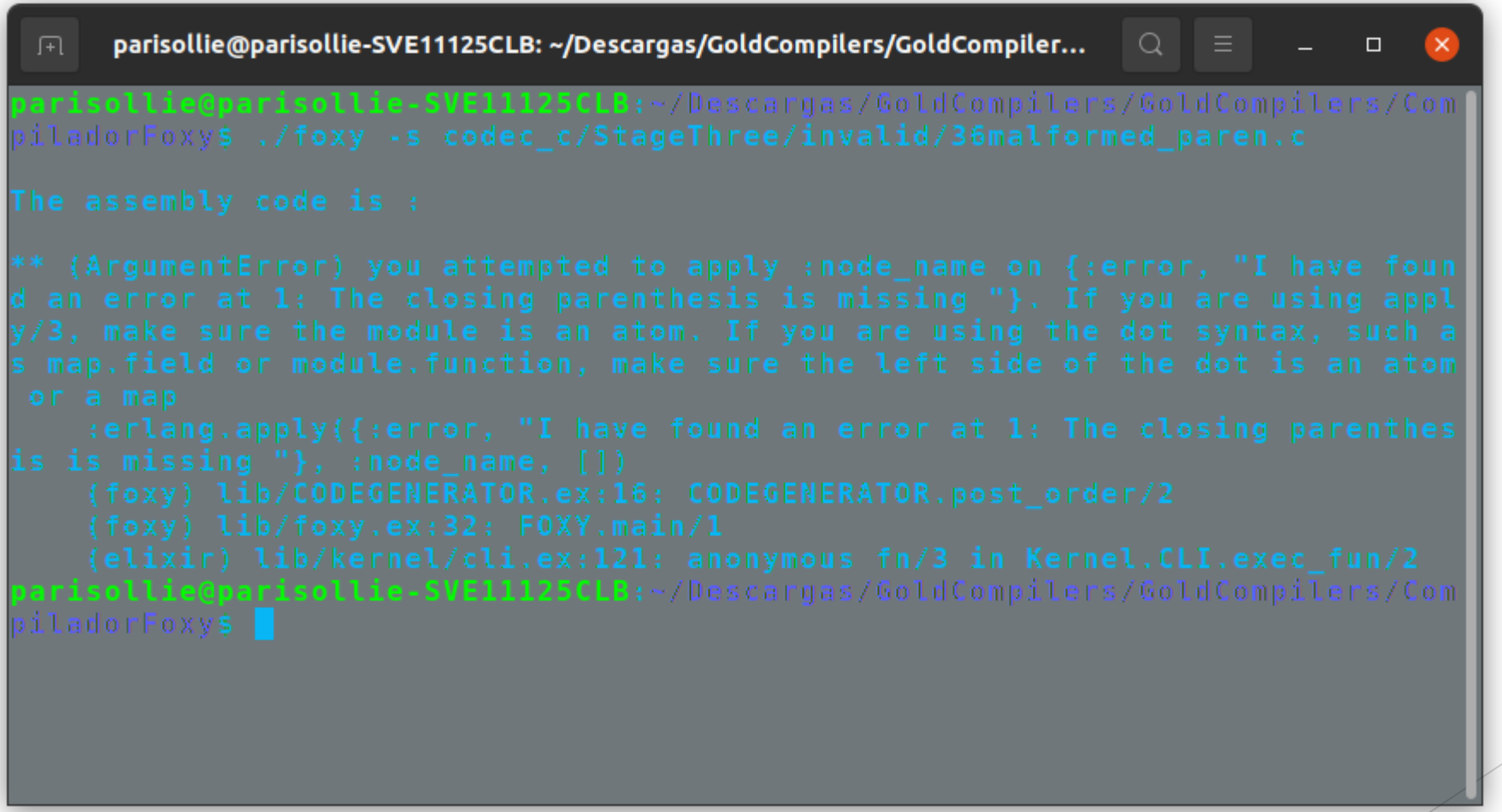


echo %errorlevel%
check the return code; it should be 4

```
C:\Users\Henry\Desktop\CompiladorFoxy\codec_c\StageThree\Valid>24add.exe

C:\Users\Henry\Desktop\CompiladorFoxy\codec_c\StageThree\Valid>echo %errorlevel%
4
```

Test Invalid:



Plan work:

Work plan: Gold Gompilers																	
Task				February				July								August	
				Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4					Week 1	Week 2
Requirements.																	
Client Requirements.																	
Read: Nora Sandler document.																	
Install elixir and read documentation.																	
First team meeting.																	
Stage 1: Lexer.																	
Project architecture definition.																	
Lexer code.																	
Stage 2: Parser.																	
Second team meeting.																	
Parser Code.																	
Stage 3: Code Generator.																	
x64 Instructions Set Review.																	
Code generator.																	
Stage 4: Tests.																	
Test building.																	
Final team meeting.																	
Documentation.																	
Deliverable.																	

- Thank you !