Stage 1: Integers

Team Assembly

Project Manager

Experiences

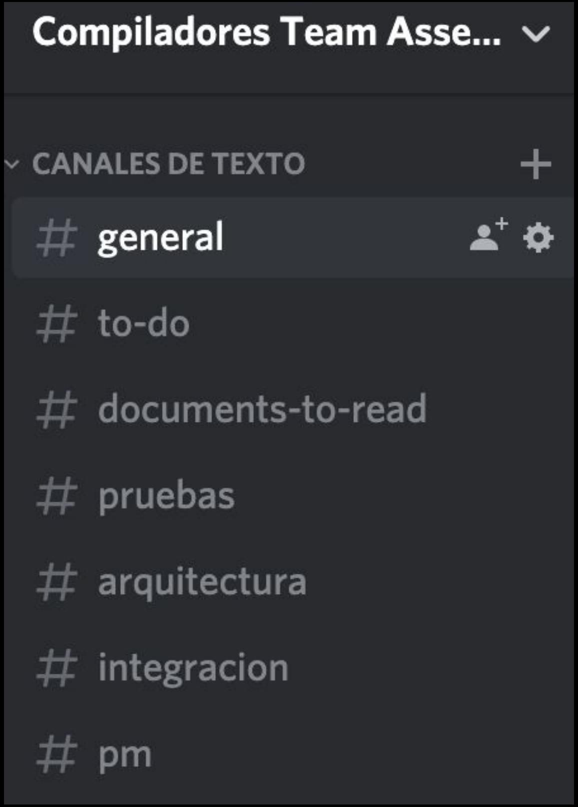
Experiences

elixir

Project Manager

Way of working

Way of Working


Compiladores Team Asse...

CANALES DE TEXTO

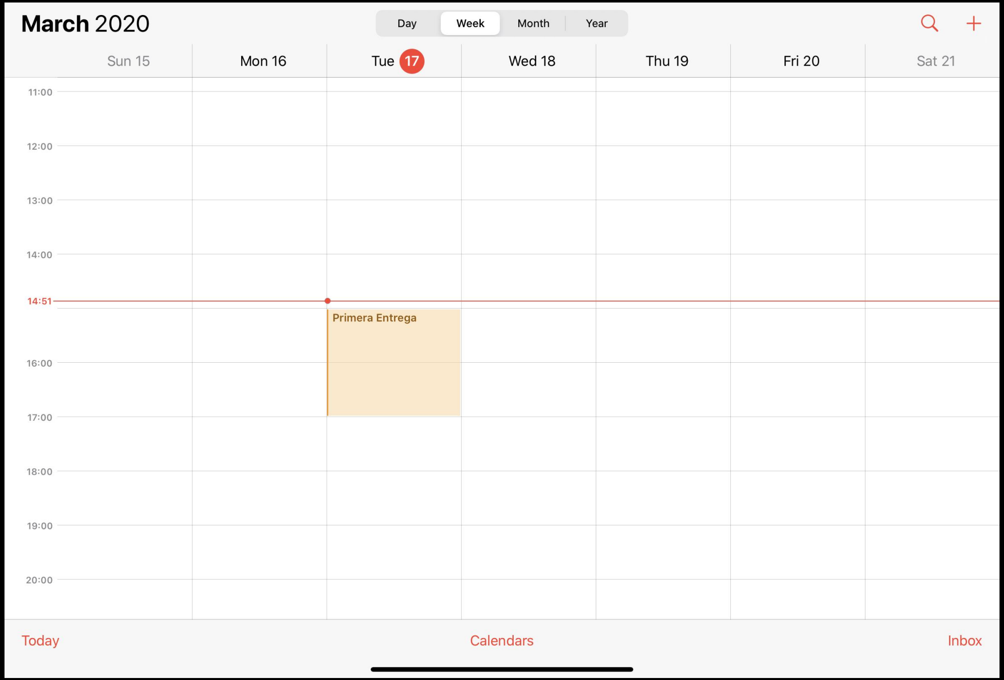
- # general
- # to-do
- # documents-to-read
- # pruebas
- # arquitectura
- # integracion
- # pm

Way of Working

Friday

29

zoom



Architect

- Experiences
    - Implementing the Data Structures/Algorithms in paper
    - Explaining the requirements to the team
    - Helping when something doesn’t work

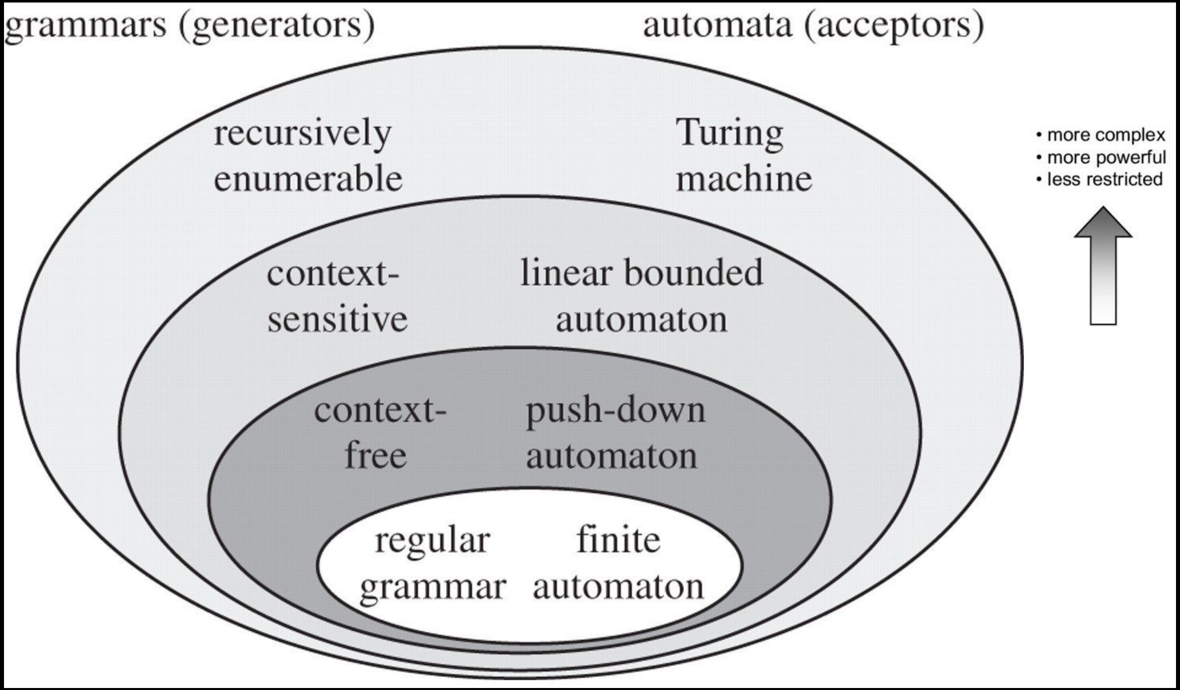
- Way of working
    - Logical implementation of algorithms
    - Lots of self-criticism to find edge cases
    - Constantly evaluating team’s code

System Architecture: Idea

Goal: Code Less, Think More

- How to implement a flexible system that saves us work in the future?
- How to stop ourselves from going back and changing our code?
- How to streamline future expansions and development?

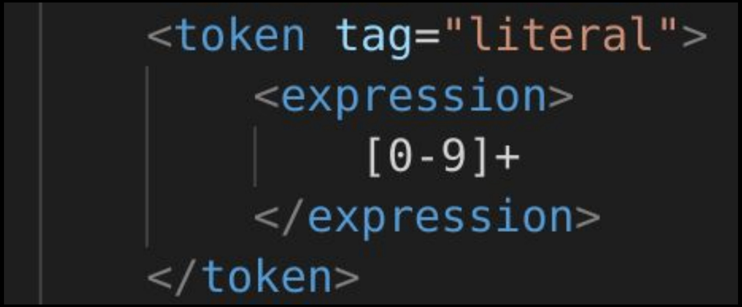
System Architecture: Solution - Abstraction



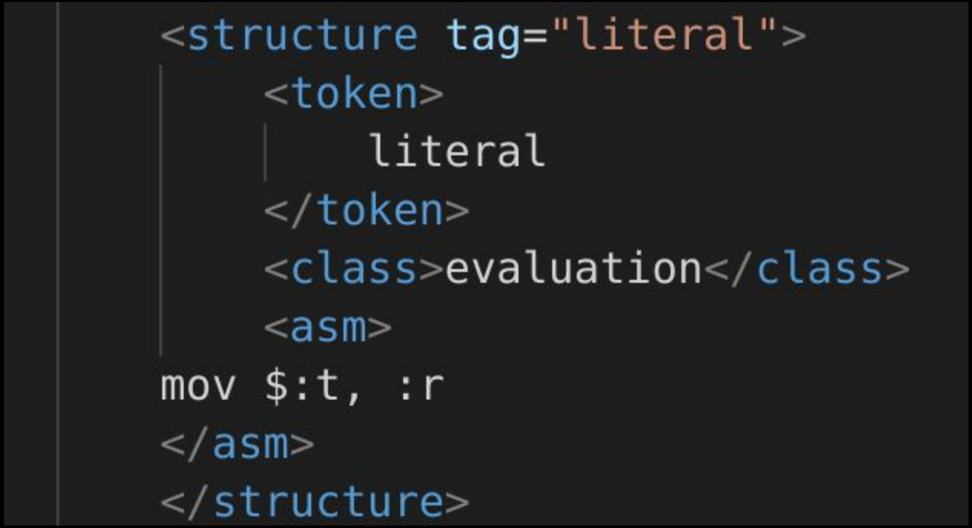
How to bring this abstraction down to Earth?

XML Files - Easy to parse for both humans and computers.

Logical consistency in Grammar/Alphabet makes for a strong system.



```
<token tag="literal">
    <expression>
        [0-9]+
    </expression>
</token>
```

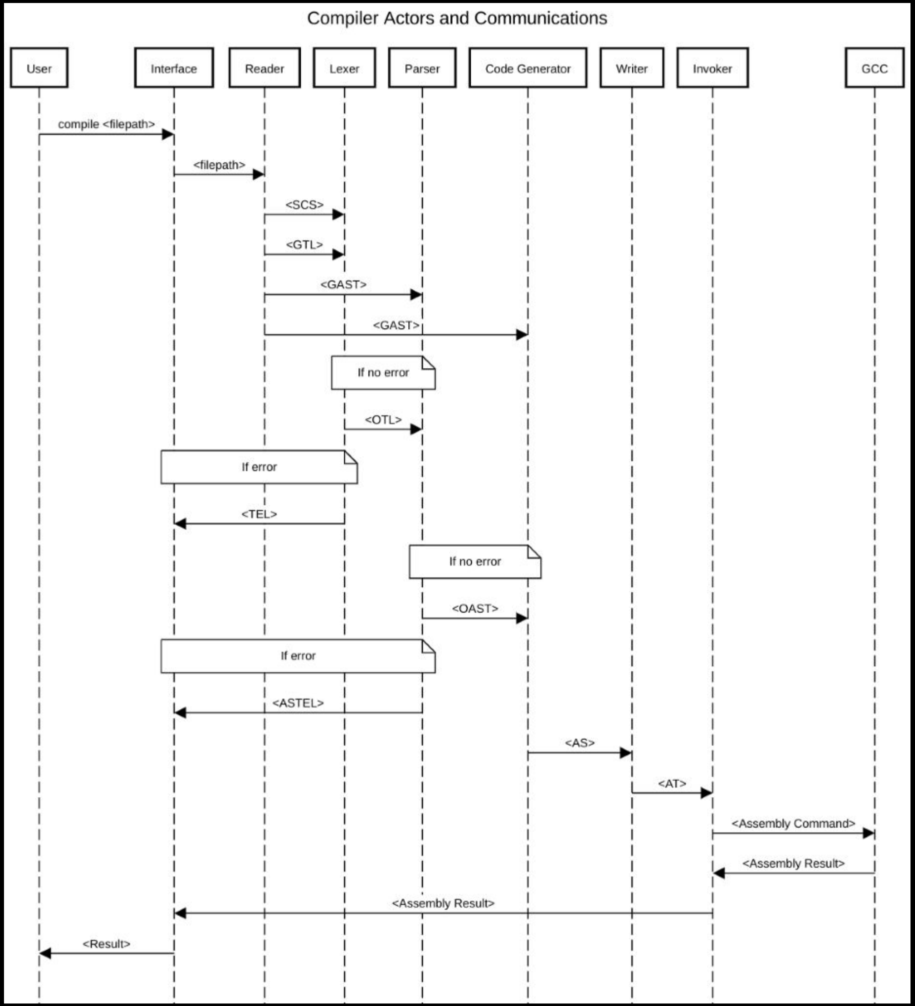


```
<structure tag="literal">
    <token>
        literal
    </token>
    <class>evaluation</class>
    <asm>
mov $:t, :r
</asm>
</structure>
```

Implementation

It’s complicated…

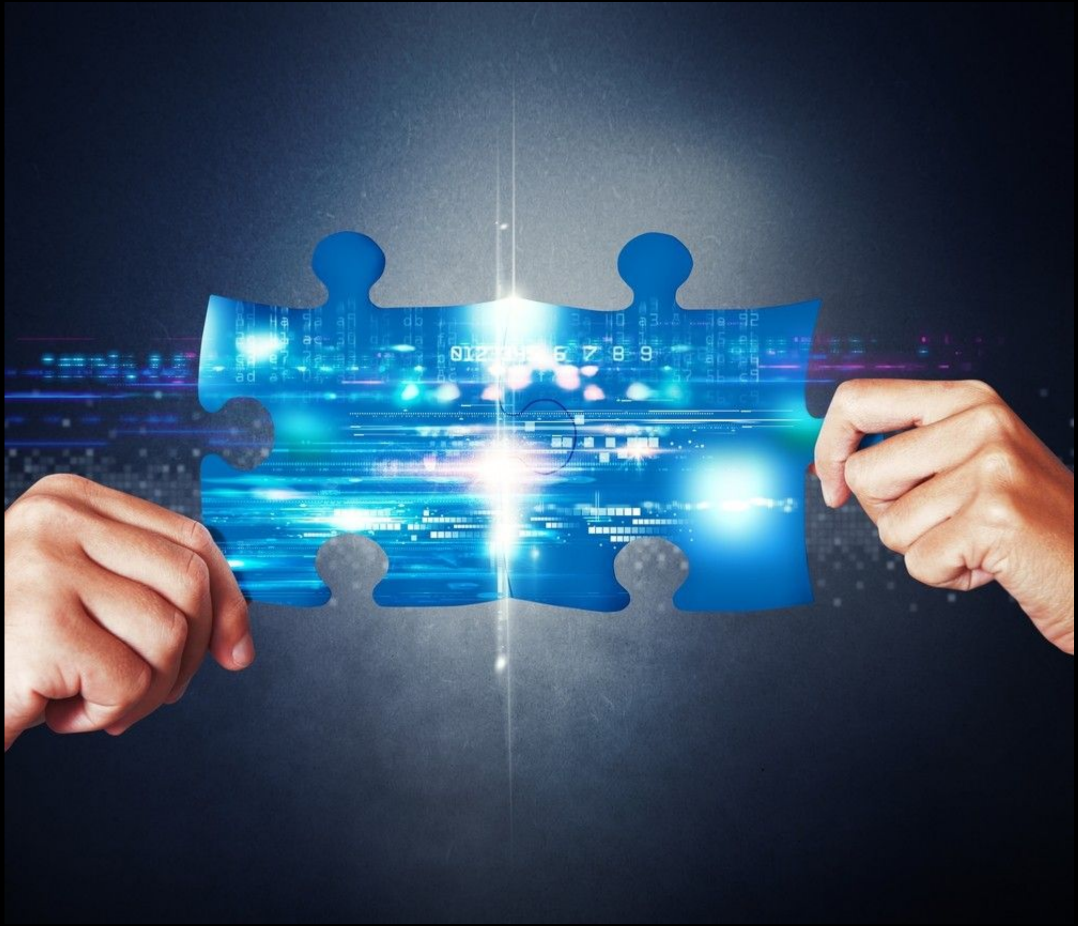
- Actors with outputs/inputs
- Data Structures sent around
- Modularization of the system



Integrator

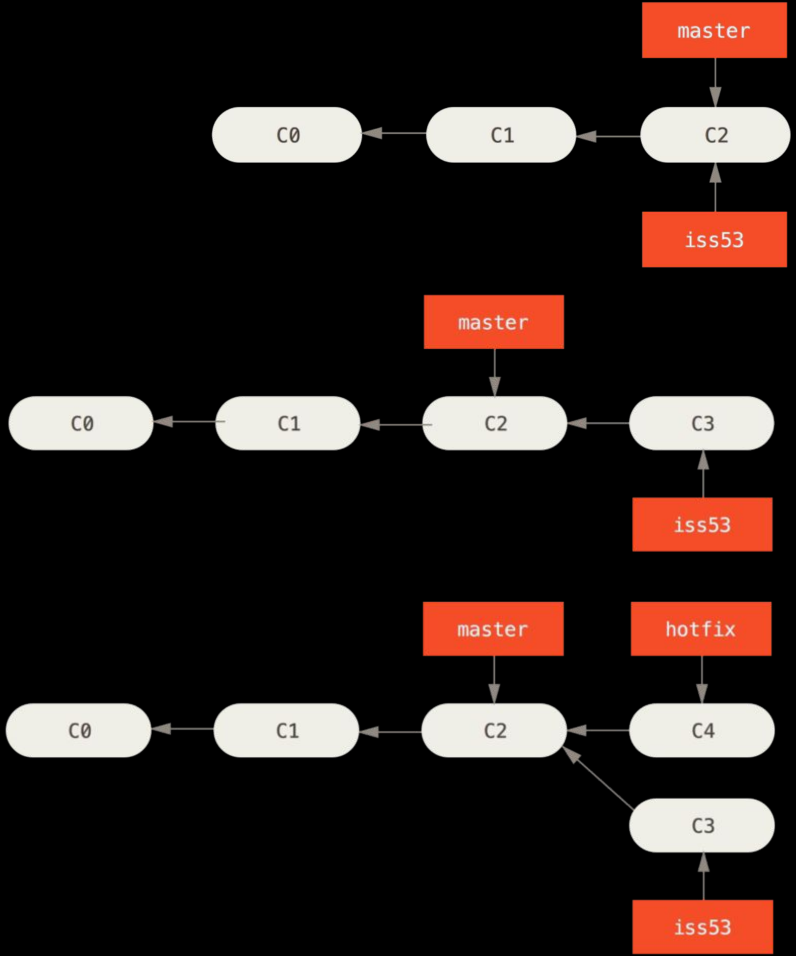
- Experiences
  - Technical part
  - With team
  - Other

- Way of working
  - Use of Windows and Linux
  - Use of git
  - Start point
  - Risk

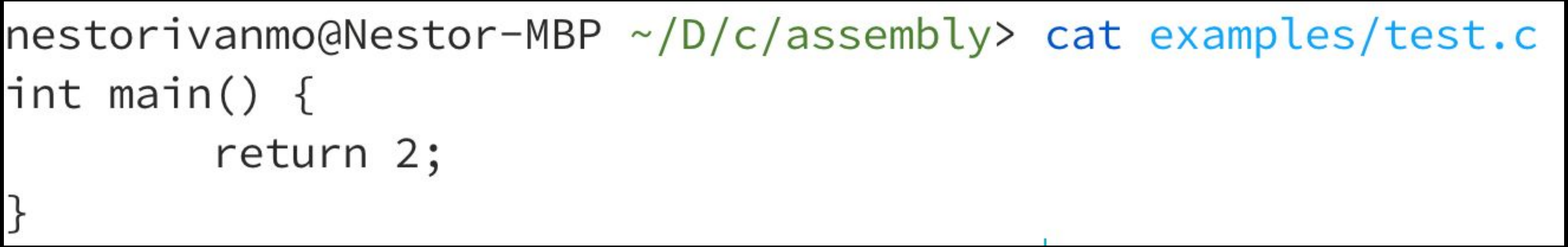


Integrations

- Reader
- Lexer
- Parser
- Code generator
- Writer
- Invoker

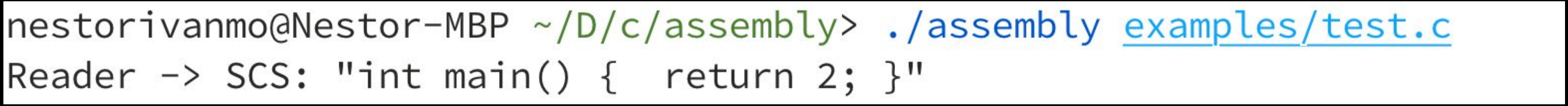


Assembly Compiler - Valid Test 1

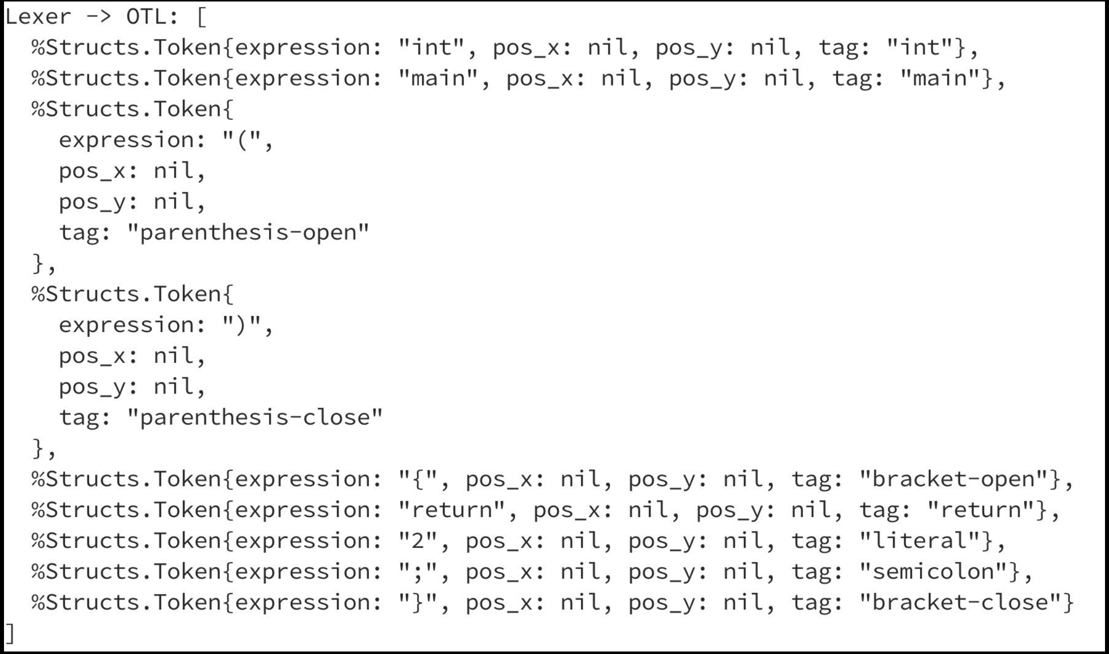


```
nestorivanmo@Nestor-MBP ~/D/c/assembly> cat examples/test.c
int main() {
        return 2;
}
```

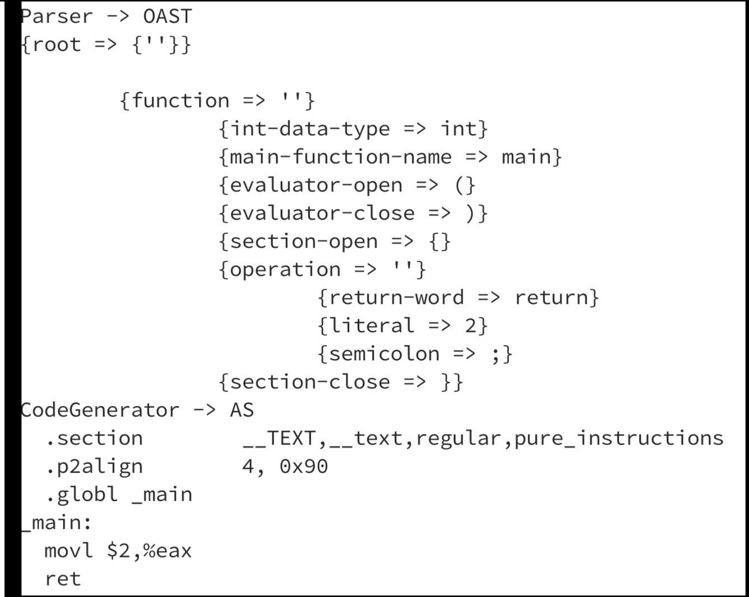
Assembly Compiler - Output for Test 1



```
nestorivanmo@Nestor-MBP ~/D/c/assembly> ./assembly examples/test.c
Reader -> SCS: "int main() {  return 2; }"
```

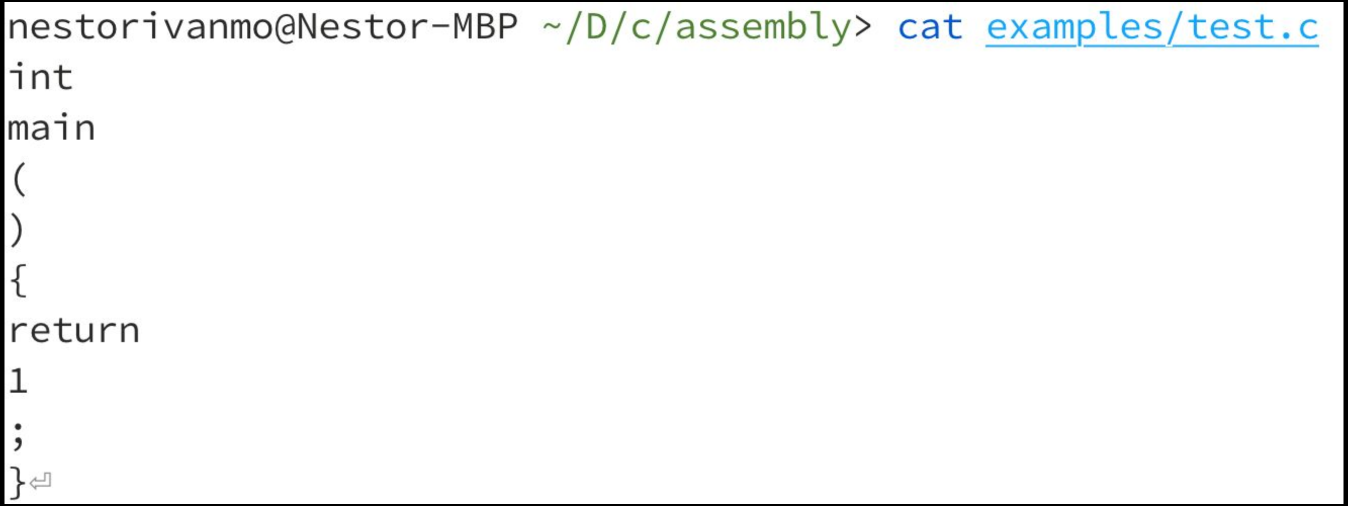


```
Lexer -> OTL: [
  %Structs.Token{expression: "int", pos_x: nil, pos_y: nil, tag: "int"},
  %Structs.Token{expression: "main", pos_x: nil, pos_y: nil, tag: "main"},
  %Structs.Token{
    expression: "(",
    pos_x: nil,
    pos_y: nil,
    tag: "parenthesis-open"
  },
  %Structs.Token{
    expression: ")",
    pos_x: nil,
    pos_y: nil,
    tag: "parenthesis-close"
  },
  %Structs.Token{expression: "{", pos_x: nil, pos_y: nil, tag: "bracket-open"},
  %Structs.Token{expression: "return", pos_x: nil, pos_y: nil, tag: "return"},
  %Structs.Token{expression: "2", pos_x: nil, pos_y: nil, tag: "literal"},
  %Structs.Token{expression: ";", pos_x: nil, pos_y: nil, tag: "semicolon"},
  %Structs.Token{expression: "}", pos_x: nil, pos_y: nil, tag: "bracket-close"}
]
```



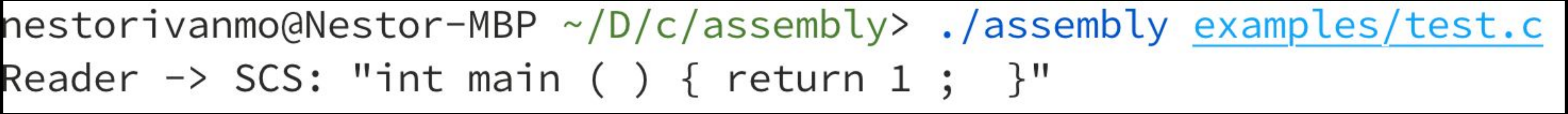
```
Parser -> OAST
{root => {''}}

        {function => ''}
                {int-data-type => int}
                {main-function-name => main}
                {evaluator-open => (}
                {evaluator-close => )}
                {section-open => {}
                {operation => ''}
                        {return-word => return}
                        {literal => 2}
                        {semicolon => ;}
                {section-close => }}
CodeGenerator -> AS
  .section        __TEXT,__text,regular,pure_instructions
  .p2align        4, 0x90
  .globl _main
_main:
  movl $2,%eax
  ret
```

Assembly Compiler - Valid Test 2

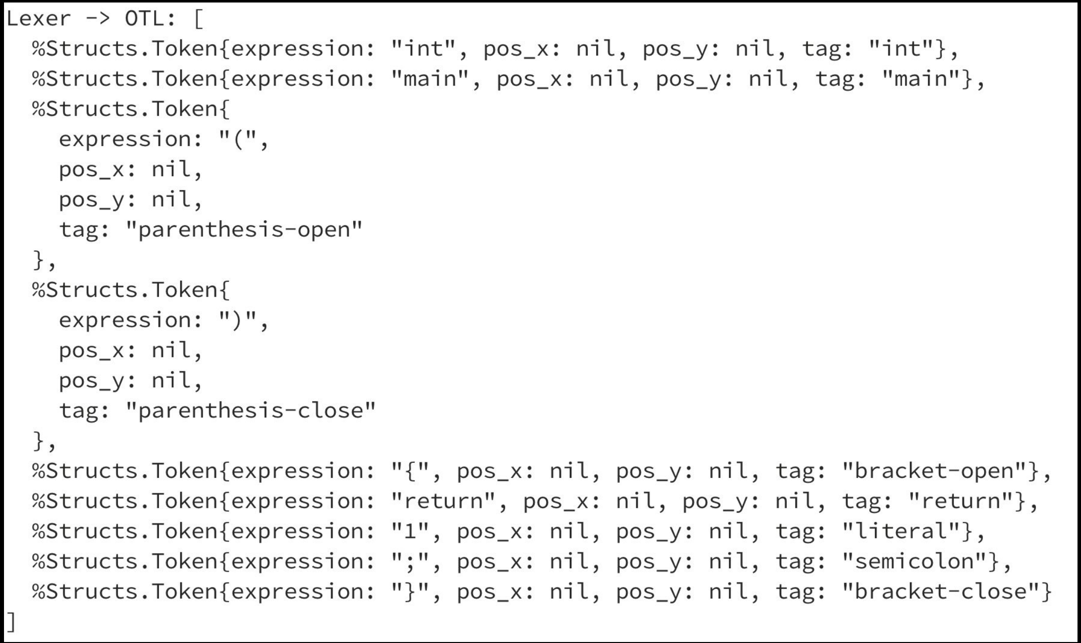


```
nestorivanmo@Nestor-MBP ~/D/c/assembly> cat examples/test.c
int
main
(
)
{
return
1
;
}⏎
```

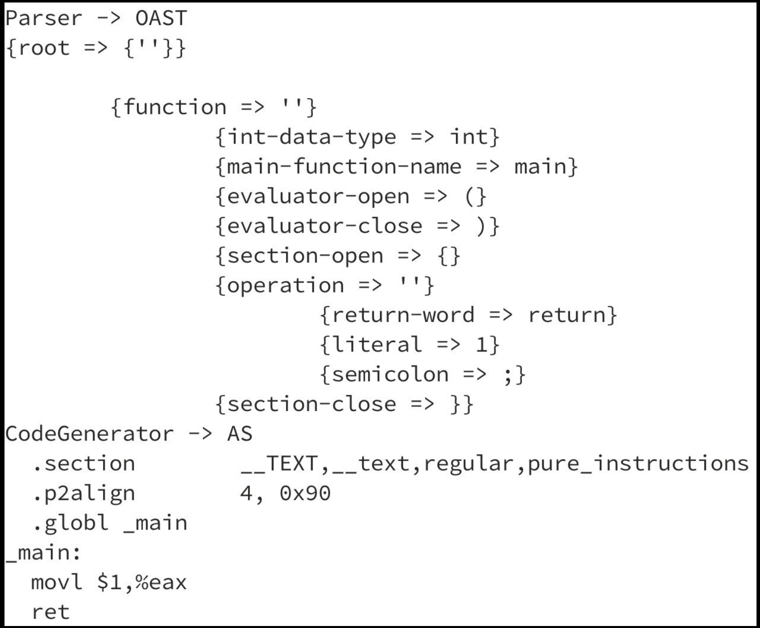
Assembly Compiler - Output for Valid Test 2



```
nestorivanmo@Nestor-MBP ~/D/c/assembly> ./assembly examples/test.c
Reader -> SCS: "int main ( ) { return 1 ;  }"
```

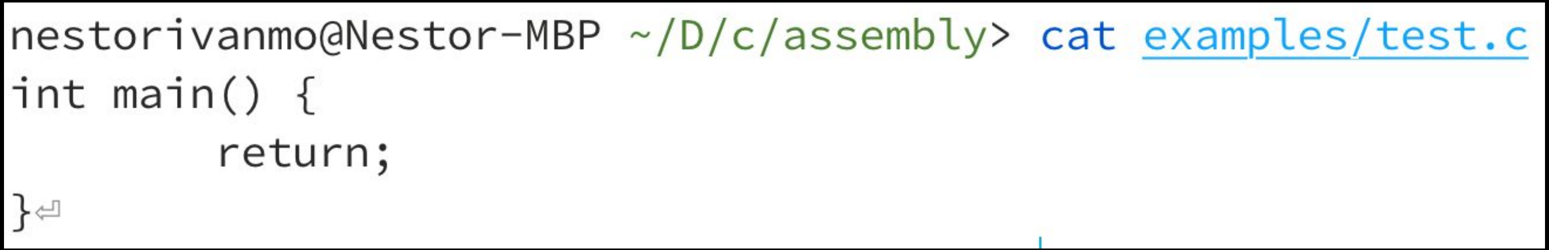


```
Lexer -> OTL: [
  %Structs.Token{expression: "int", pos_x: nil, pos_y: nil, tag: "int"},
  %Structs.Token{expression: "main", pos_x: nil, pos_y: nil, tag: "main"},
  %Structs.Token{
    expression: "(",
    pos_x: nil,
    pos_y: nil,
    tag: "parenthesis-open"
  },
  %Structs.Token{
    expression: ")",
    pos_x: nil,
    pos_y: nil,
    tag: "parenthesis-close"
  },
  %Structs.Token{expression: "{", pos_x: nil, pos_y: nil, tag: "bracket-open"},
  %Structs.Token{expression: "return", pos_x: nil, pos_y: nil, tag: "return"},
  %Structs.Token{expression: "1", pos_x: nil, pos_y: nil, tag: "literal"},
  %Structs.Token{expression: ";", pos_x: nil, pos_y: nil, tag: "semicolon"},
  %Structs.Token{expression: "}", pos_x: nil, pos_y: nil, tag: "bracket-close"}
]
```



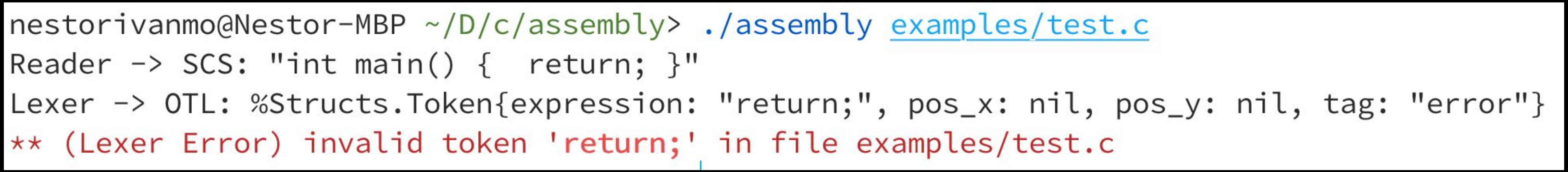
```
Parser -> OAST
{root => {''}}

        {function => ''}
                {int-data-type => int}
                {main-function-name => main}
                {evaluator-open => (}
                {evaluator-close => )}
                {section-open => {}
                {operation => ''}
                        {return-word => return}
                        {literal => 1}
                        {semicolon => ;}
                {section-close => }}
CodeGenerator -> AS
  .section        __TEXT,__text,regular,pure_instructions
  .p2align        4, 0x90
  .globl _main
_main:
  movl $1,%eax
  ret
```

Assembly Compiler - Invalid Test 3



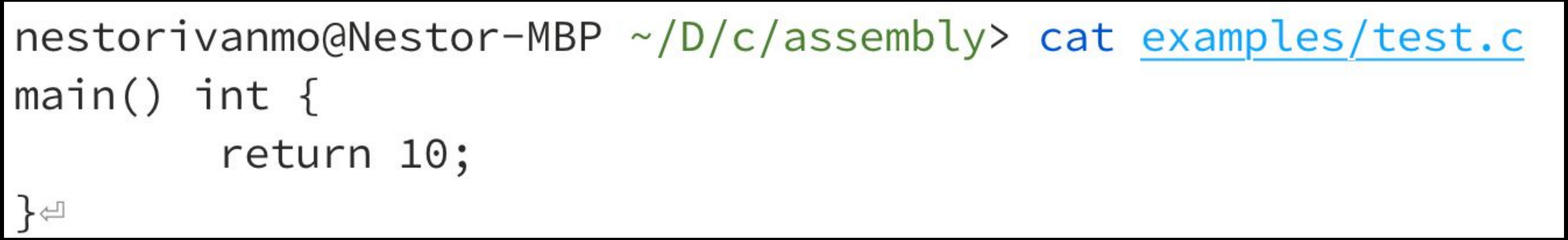
```
nestorivanmo@Nestor-MBP ~/D/c/assembly> cat examples/test.c
int main() {
        return;
}⏎
```

Assembly Compiler - Output for Invalid Test 3



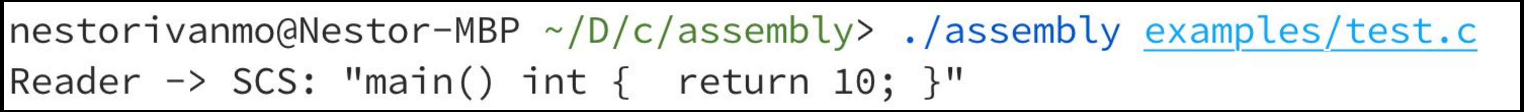
```
nestorivanmo@Nestor-MBP ~/D/c/assembly> ./assembly examples/test.c
Reader -> SCS: "int main() {  return; }"
Lexer -> OTL: %Structs.Token{expression: "return;", pos_x: nil, pos_y: nil, tag: "error"}
** (Lexer Error) invalid token 'return;' in file examples/test.c
```

Assembly Compiler - Invalid Test 4

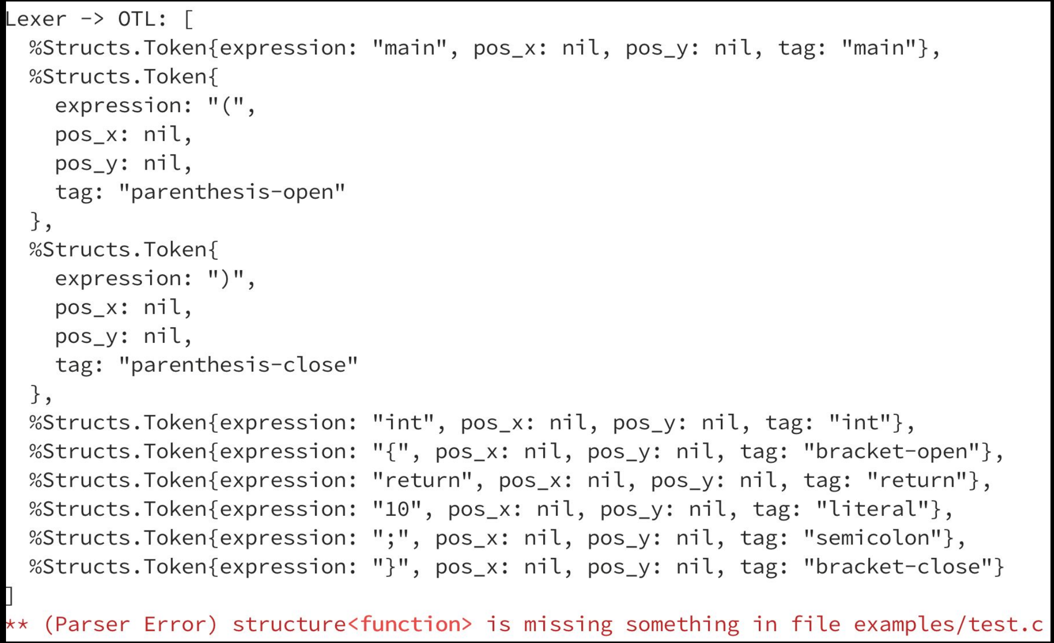


```
nestorivanmo@Nestor-MBP ~/D/c/assembly> cat examples/test.c
main() int {
        return 10;
}⏎
```

Assembly Compiler - Output for Invalid Test 4

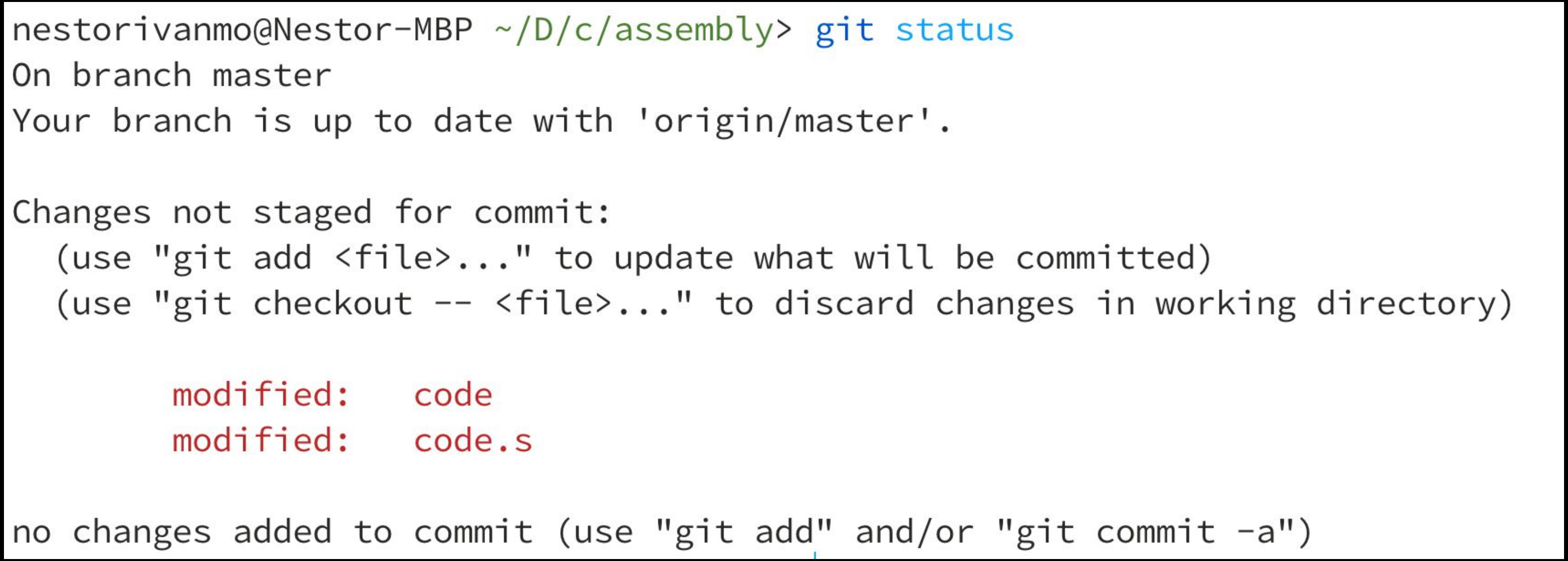


```
nestorivanmo@Nestor-MBP ~/D/c/assembly> ./assembly examples/test.c
Reader -> SCS: "main() int {  return 10; }"
```



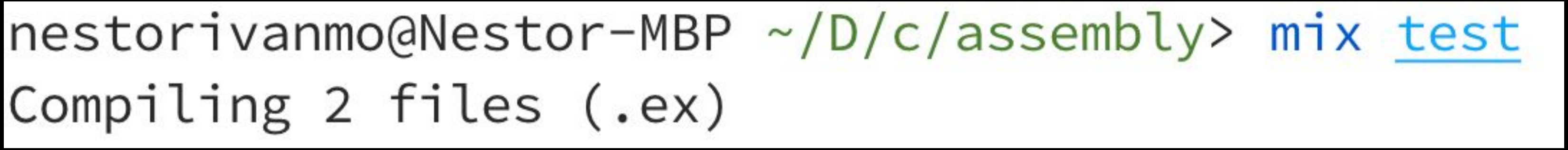
```
Lexer -> OTL: [
  %Structs.Token{expression: "main", pos_x: nil, pos_y: nil, tag: "main"},
  %Structs.Token{
    expression: "(",
    pos_x: nil,
    pos_y: nil,
    tag: "parenthesis-open"
  },
  %Structs.Token{
    expression: ")",
    pos_x: nil,
    pos_y: nil,
    tag: "parenthesis-close"
  },
  %Structs.Token{expression: "int", pos_x: nil, pos_y: nil, tag: "int"},
  %Structs.Token{expression: "{", pos_x: nil, pos_y: nil, tag: "bracket-open"},
  %Structs.Token{expression: "return", pos_x: nil, pos_y: nil, tag: "return"},
  %Structs.Token{expression: "10", pos_x: nil, pos_y: nil, tag: "literal"},
  %Structs.Token{expression: ";", pos_x: nil, pos_y: nil, tag: "semicolon"},
  %Structs.Token{expression: "}", pos_x: nil, pos_y: nil, tag: "bracket-close"}
]
** (Parser Error) structure<function> is missing something in file examples/test.c
```

Compiler outputs

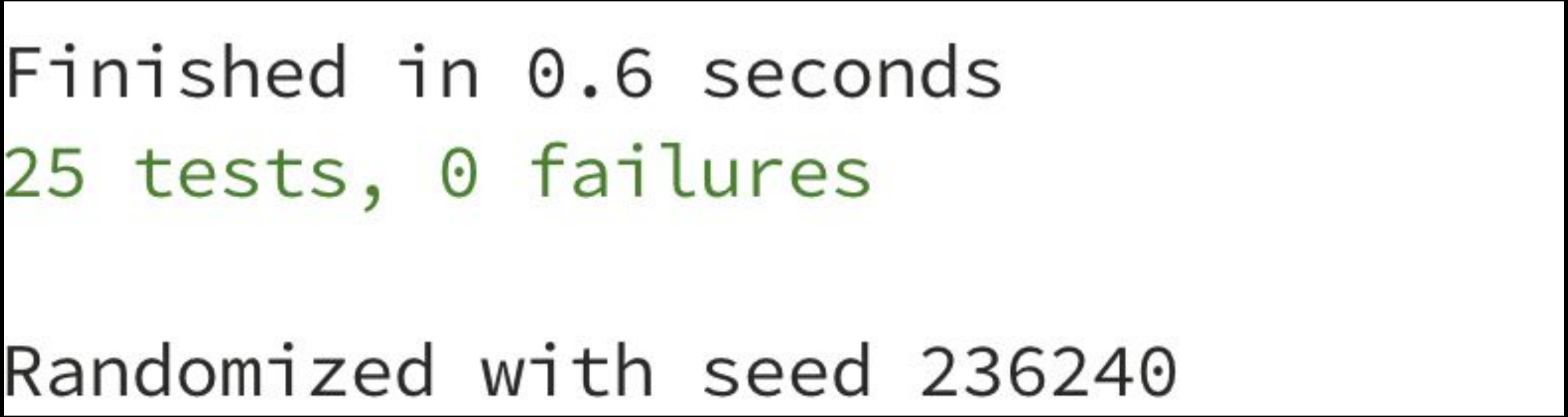


```
nestorivanmo@Nestor-MBP ~/D/c/assembly> git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   code
        modified:   code.s

no changes added to commit (use "git add" and/or "git commit -a")
```

Compiler tests

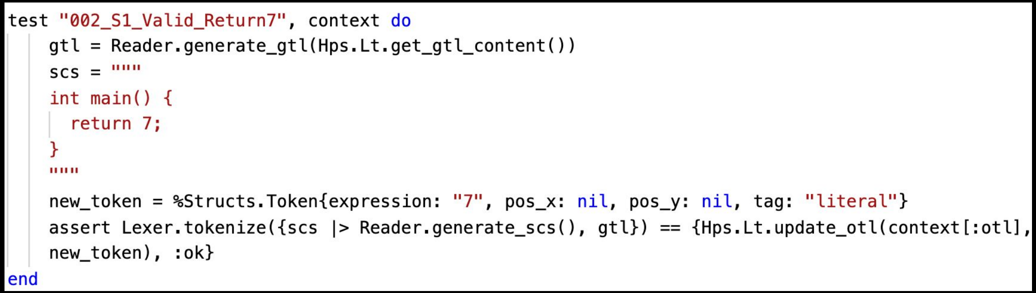


```
nestorivanmo@Nestor-MBP ~/D/c/assembly> mix test
Compiling 2 files (.ex)
```

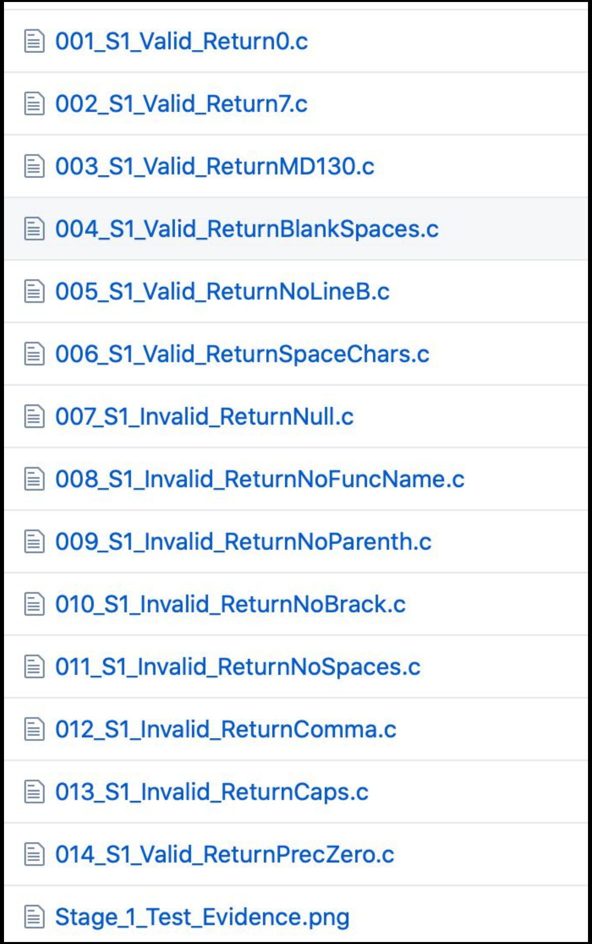


```
Finished in 0.6 seconds
25 tests, 0 failures


Randomized with seed 236240
```

Tester



```
test "002_S1_Valid_Return7", context do
    gtl = Reader.generate_gtl(Hps.Lt.get_gtl_content())
    scs = """
    int main() {
      return 7;
    }
    """
    new_token = %Structs.Token{expression: "7", pos_x: nil, pos_y: nil, tag: "literal"}
    assert Lexer.tokenize({scs |> Reader.generate_scs(), gtl}) == {Hps.Lt.update_otl(context[:otl],
    new_token), :ok}
end
```



- 001_S1_Valid_Return0.c
- 002_S1_Valid_Return7.c
- 003_S1_Valid_ReturnMD130.c
- 004_S1_Valid_ReturnBlankSpaces.c
- 005_S1_Valid_ReturnNoLineB.c
- 006_S1_Valid_ReturnSpaceChars.c
- 007_S1_Invalid_ReturnNull.c
- 008_S1_Invalid_ReturnNoFuncName.c
- 009_S1_Invalid_ReturnNoParenth.c
- 010_S1_Invalid_ReturnNoBrack.c
- 011_S1_Invalid_ReturnNoSpaces.c
- 012_S1_Invalid_ReturnComma.c
- 013_S1_Invalid_ReturnCaps.c
- 014_S1_Valid_ReturnPrecZero.c
- Stage_1_Test_Evidence.png

References

https://norasandler.com

https://www.amazon.com/Engineering-Compiler-Keith-Cooper/dp/012088478X

https://elixir-lang.org

https://hexdocs.pm/elixir/Enum.html

https://git-scm.com/book/es/v2/Ramificaciones-en-Git-Procedimientos-B%C3%A1sicos-para-Ramificar-y-Fusionar