Final Delivery

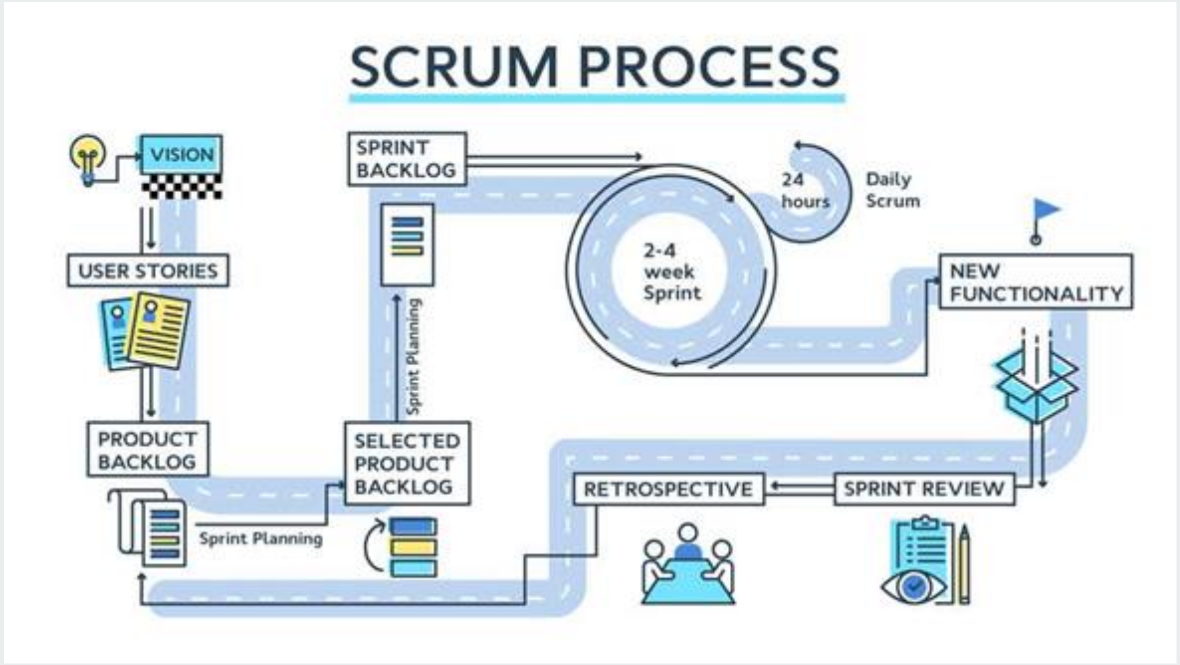
Barrientos Veana Luis Mauricio.

González Pacheco Leonardo Alonso.

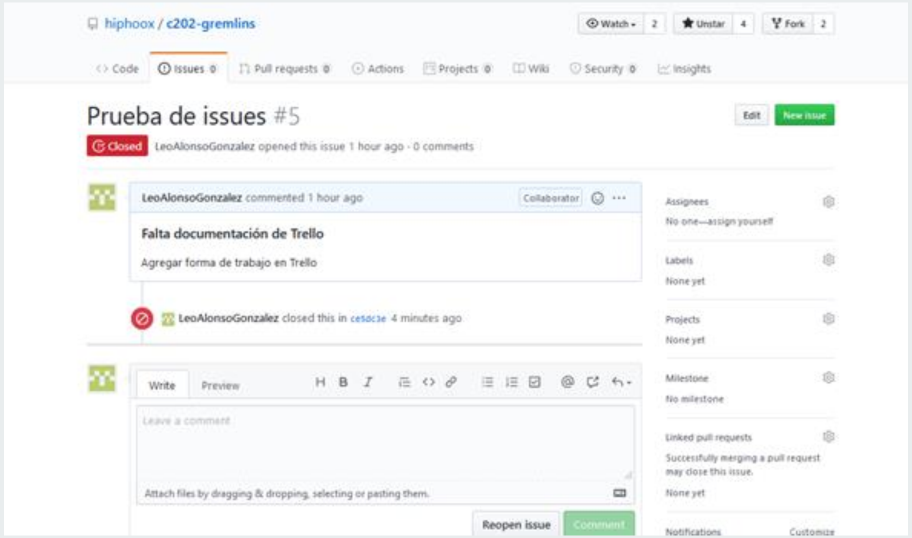
Martínez Matías Joan Eduardo.

Rosales Romero Ricardo.

SCRUM


SCRUM PROCESS
VISION
USER STORIES
PRODUCT BACKLOG
Sprint Planning
SELECTED PRODUCT BACKLOG
Sprint Planning
SPRINT BACKLOG
2-4 week Sprint
24 hours
Daily Scrum
NEW FUNCTIONALITY
SPRINT REVIEW
RETROSPECTIVE

Learnings on github



hiphoox / c202-gremlins

Watch 2 · Unstar 4 · Fork 2

Code · Issues 0 · Pull requests 0 · Actions · Projects 0 · Wiki · Security 0 · Insights

Prueba de issues #5

Edit · New issue

Closed LeoAlonsoGonzalez opened this issue 1 hour ago · 0 comments

LeoAlonsoGonzalez commented 1 hour ago · Collaborator

Falta documentación de Trello

Agregar forma de trabajo en Trello

LeoAlonsoGonzalez closed this in ce5dc3e 4 minutes ago

Write · Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Reopen issue · Comment

Assignees: No one—assign yourself

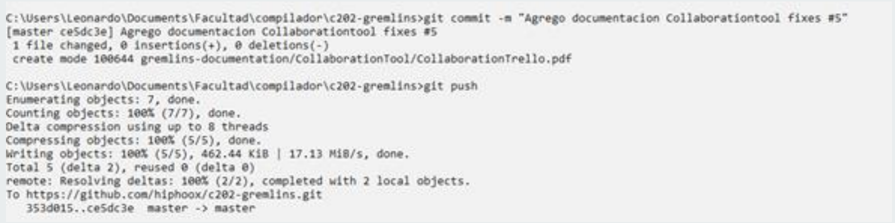
Labels: None yet

Projects: None yet

Milestone: No milestone

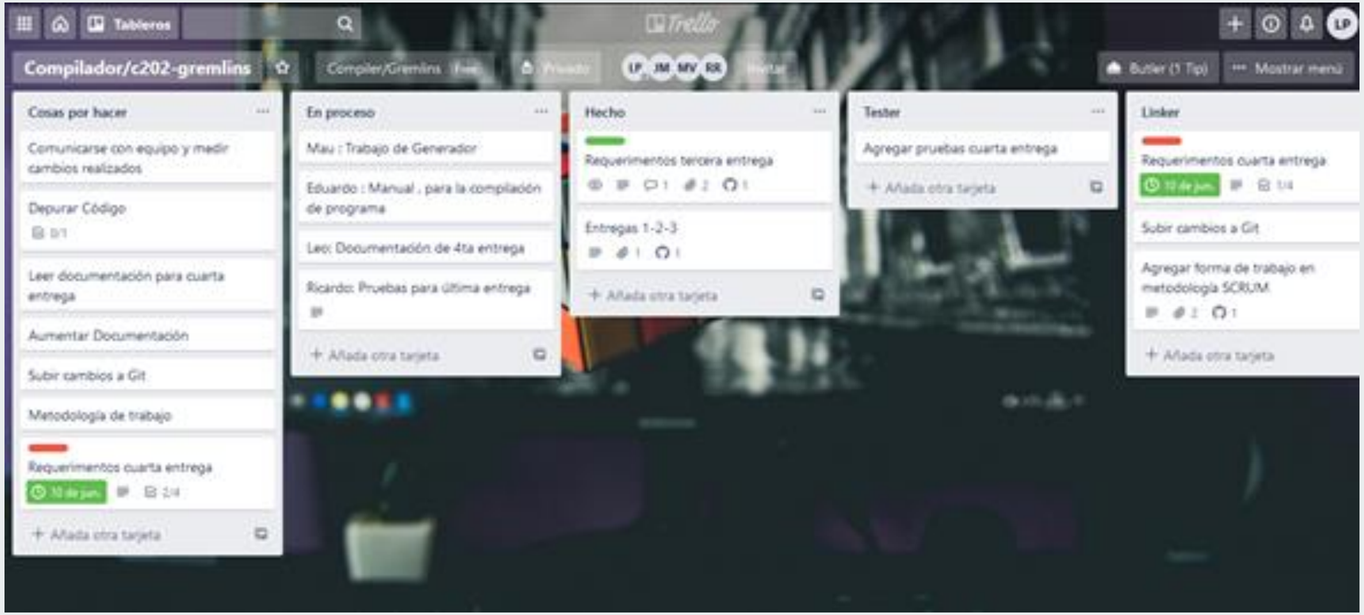
Linked pull requests: Successfully merging a pull request may close this issue. None yet

Notifications · Customize



```
C:\Users\Leonardo\Documents\Facultad\compilador\c202-gremlins>git commit -m "Agrego documentacion Collaborationtool fixes #5"
[master ce5dc3e] Agrego documentacion Collaborationtool fixes #5
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 gremlins-documentation/CollaborationTool/CollaborationTrello.pdf

C:\Users\Leonardo\Documents\Facultad\compilador\c202-gremlins>git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 462.44 KiB | 17.13 MiB/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/hiphoox/c202-gremlins.git
   353d015..ce5dc3e  master -> master
```

Collaboration tools



Cosas por hacer

- Comunicarse con equipo y medir cambios realizados
- Depurar Código 0/1
- Leer documentación para cuarta entrega
- Aumentar Documentación
- Subir cambios a Git
- Metodología de trabajo
- Requerimientos cuarta entrega 10 de jun. 2/4
- + Añada otra tarjeta

En proceso

- Mau : Trabajo de Generador
- Eduardo : Manual , para la compilación de programa
- Leo: Documentación de 4ta entrega
- Ricardo: Pruebas para última entrega
- + Añada otra tarjeta

Hecho

- Requerimientos tercera entrega 1 2 1
- Entregas 1-2-3 1 1
- + Añada otra tarjeta

Tester

- Agregar pruebas cuarta entrega
- + Añada otra tarjeta

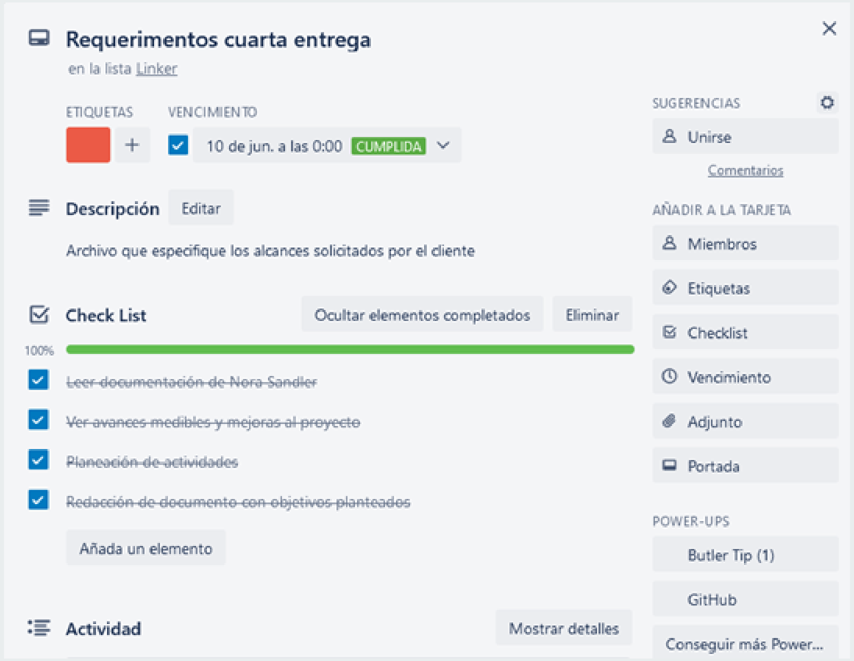
Linker

- Requerimientos cuarta entrega 10 de jun. 1/4
- Subir cambios a Git
- Agregar forma de trabajo en metodología SCRUM 2 1
- + Añada otra tarjeta

Collaboration tools Trello

Collaboration CCompiler Gremlins

https://trello.com/invite/b/aMgzIBO5/36c5c9122b2698930eb0a51f424dd55e/compilador-c202-gremlins



Requerimentos cuarta entrega

en la lista Linker

ETIQUETAS

VENCIMIENTO: ✓ 10 de jun. a las 0:00 CUMPLIDA

Descripción

Archivo que especifique los alcances solicitados por el diente

Check List

100%

- [x] ~~Leer documentación de Nora Sandler~~
- [x] ~~Ver avances medibles y mejoras al proyecto~~
- [x] ~~Planeación de actividades~~
- [x] ~~Redacción de documento con objetivos planteados~~

Actividad

Check list con actividades a realizar

General changes

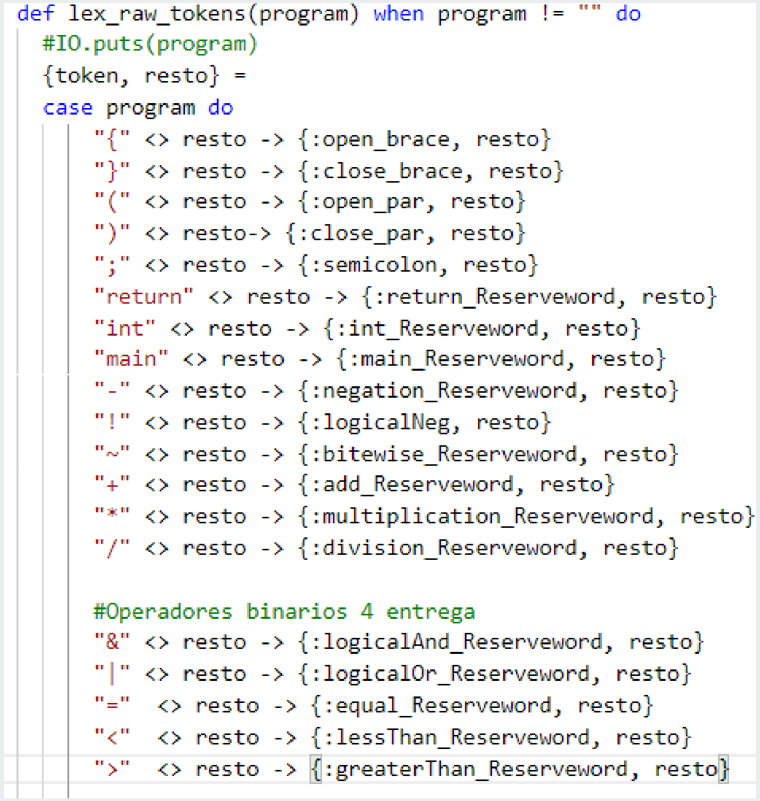
11 binary operators added.

1. Addition +
2. Multiplication *
3. Division /
4. AND &&
5. OR ||
6. Equal ==
7. Not Equal !=
8. Less than <
9. Less than or equal <=
10. Greater than >
11. Greater than or equal >=

Handle associativity and operator precedence

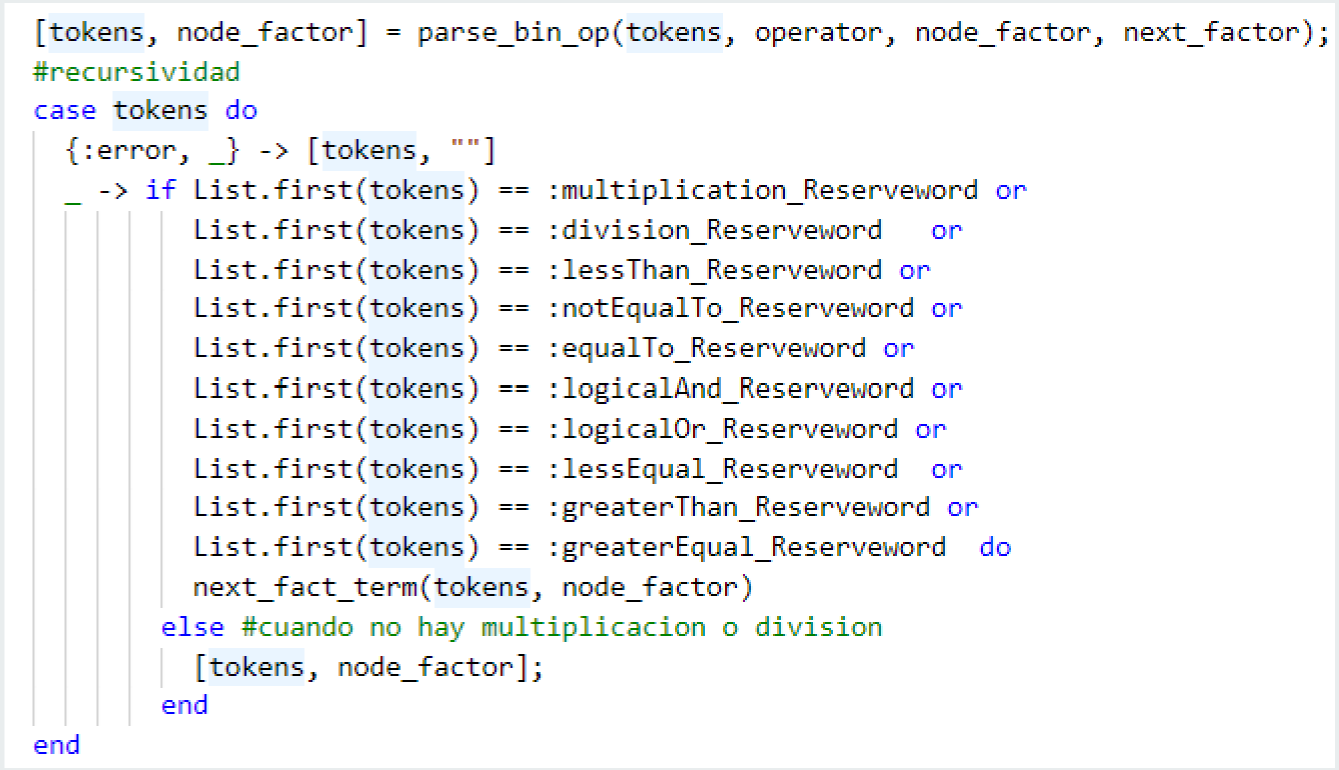
- Updates to functions in parser
- Updates to code generator

Changes in Lexer



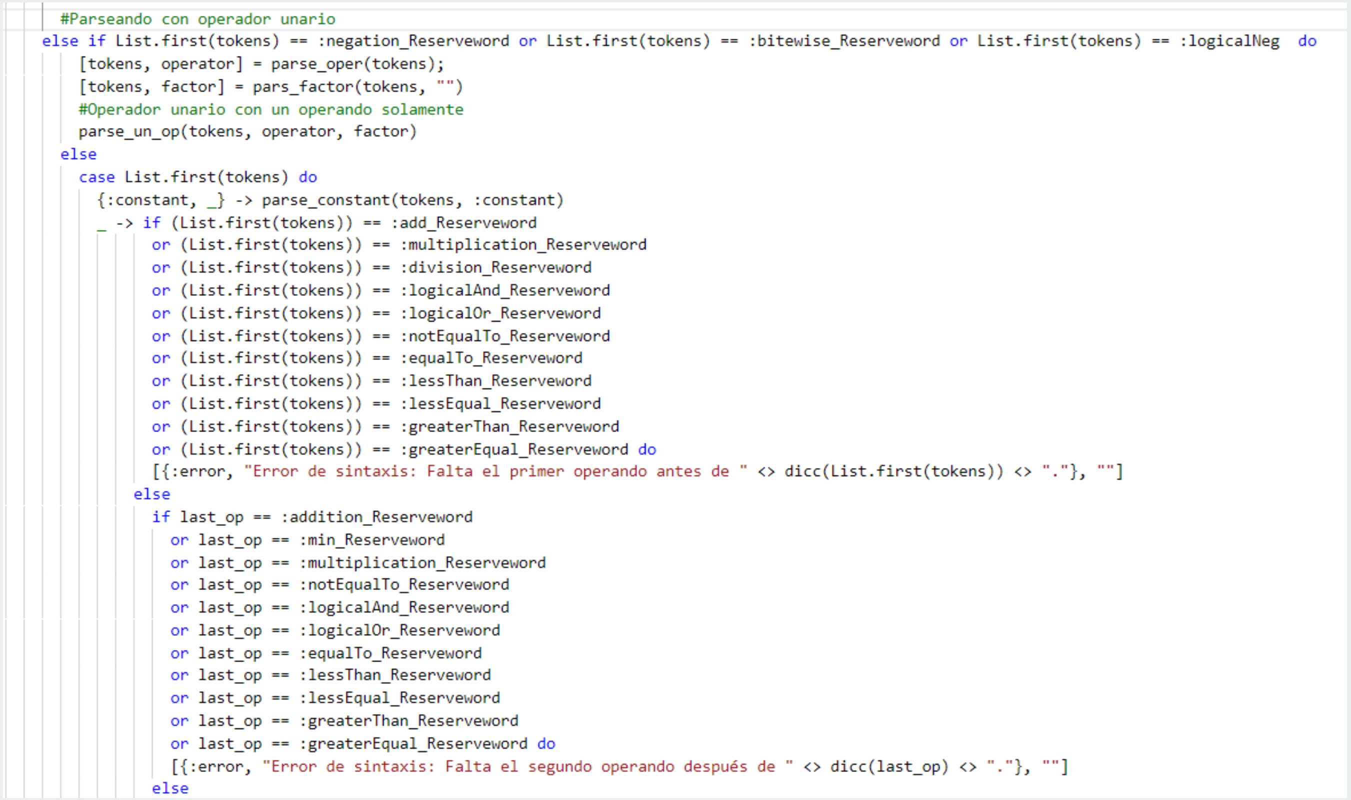
```
def lex_raw_tokens(program) when program != "" do
  #IO.puts(program)
  {token, resto} =
  case program do
      "{" <> resto -> {:open_brace, resto}
      "}" <> resto -> {:close_brace, resto}
      "(" <> resto -> {:open_par, resto}
      ")" <> resto-> {:close_par, resto}
      ";" <> resto -> {:semicolon, resto}
      "return" <> resto -> {:return_Reserveword, resto}
      "int" <> resto -> {:int_Reserveword, resto}
      "main" <> resto -> {:main_Reserveword, resto}
      "-" <> resto -> {:negation_Reserveword, resto}
      "!" <> resto -> {:logicalNeg, resto}
      "~" <> resto -> {:bitewise_Reserveword, resto}
      "+" <> resto -> {:add_Reserveword, resto}
      "*" <> resto -> {:multiplication_Reserveword, resto}
      "/" <> resto -> {:division_Reserveword, resto}

      #Operadores binarios 4 entrega
      "&" <> resto -> {:logicalAnd_Reserveword, resto}
      "|" <> resto -> {:logicalOr_Reserveword, resto}
      "="  <> resto -> {:equal_Reserveword, resto}
      "<"  <> resto -> {:lessThan_Reserveword, resto}
      ">"  <> resto -> {:greaterThan_Reserveword, resto}
```

Changes in Parser



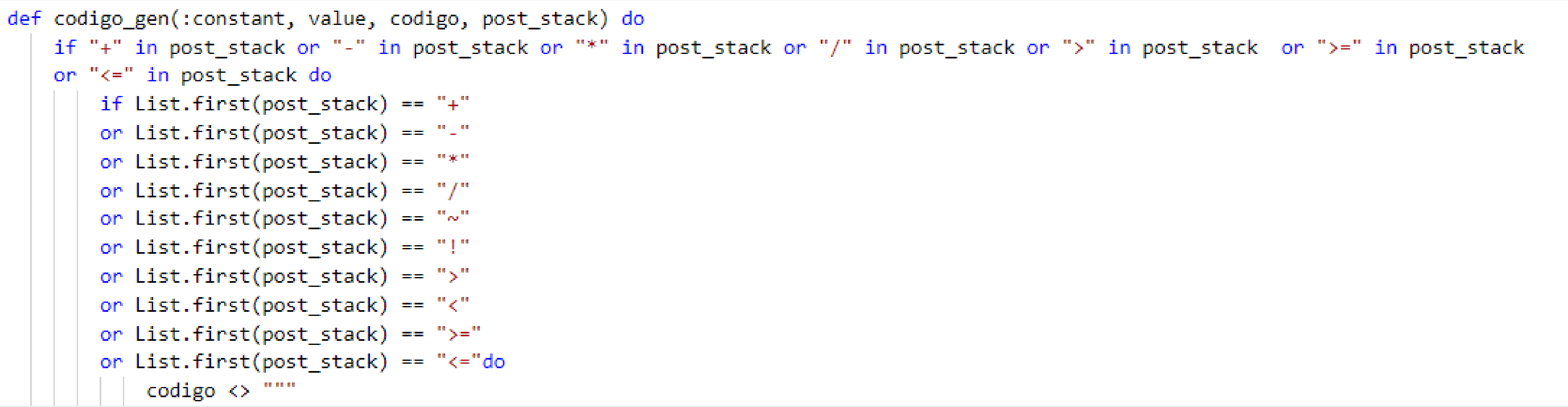
```
[tokens, node_factor] = parse_bin_op(tokens, operator, node_factor, next_factor);
#recursividad
case tokens do
  {:error, _} -> [tokens, ""]
  _ -> if List.first(tokens) == :multiplication_Reserveword or
          List.first(tokens) == :division_Reserveword   or
          List.first(tokens) == :lessThan_Reserveword or
          List.first(tokens) == :notEqualTo_Reserveword or
          List.first(tokens) == :equalTo_Reserveword or
          List.first(tokens) == :logicalAnd_Reserveword or
          List.first(tokens) == :logicalOr_Reserveword or
          List.first(tokens) == :lessEqual_Reserveword  or
          List.first(tokens) == :greaterThan_Reserveword or
          List.first(tokens) == :greaterEqual_Reserveword  do
          next_fact_term(tokens, node_factor)
        else #cuando no hay multiplicacion o division
          [tokens, node_factor];
        end
end
```

Changes in Parser



```
    #Parseando con operador unario
  else if List.first(tokens) == :negation_Reserveword or List.first(tokens) == :bitewise_Reserveword or List.first(tokens) == :logicalNeg  do
      [tokens, operator] = parse_oper(tokens);
      [tokens, factor] = pars_factor(tokens, "")
      #Operador unario con un operando solamente
      parse_un_op(tokens, operator, factor)
    else
      case List.first(tokens) do
        {:constant, _} -> parse_constant(tokens, :constant)
        _ -> if (List.first(tokens)) == :add_Reserveword
              or (List.first(tokens)) == :multiplication_Reserveword
              or (List.first(tokens)) == :division_Reserveword
              or (List.first(tokens)) == :logicalAnd_Reserveword
              or (List.first(tokens)) == :logicalOr_Reserveword
              or (List.first(tokens)) == :notEqualTo_Reserveword
              or (List.first(tokens)) == :equalTo_Reserveword
              or (List.first(tokens)) == :lessThan_Reserveword
              or (List.first(tokens)) == :lessEqual_Reserveword
              or (List.first(tokens)) == :greaterThan_Reserveword
              or (List.first(tokens)) == :greaterEqual_Reserveword do
              [{:error, "Error de sintaxis: Falta el primer operando antes de " <> dicc(List.first(tokens)) <> "."}, ""]
            else
              if last_op == :addition_Reserveword
                or last_op == :min_Reserveword
                or last_op == :multiplication_Reserveword
                or last_op == :notEqualTo_Reserveword
                or last_op == :logicalAnd_Reserveword
                or last_op == :logicalOr_Reserveword
                or last_op == :equalTo_Reserveword
                or last_op == :lessThan_Reserveword
                or last_op == :lessEqual_Reserveword
                or last_op == :greaterThan_Reserveword
                or last_op == :greaterEqual_Reserveword do
                [{:error, "Error de sintaxis: Falta el segundo operando después de " <> dicc(last_op) <> "."}, ""]
              else
```

Handling binary expressions



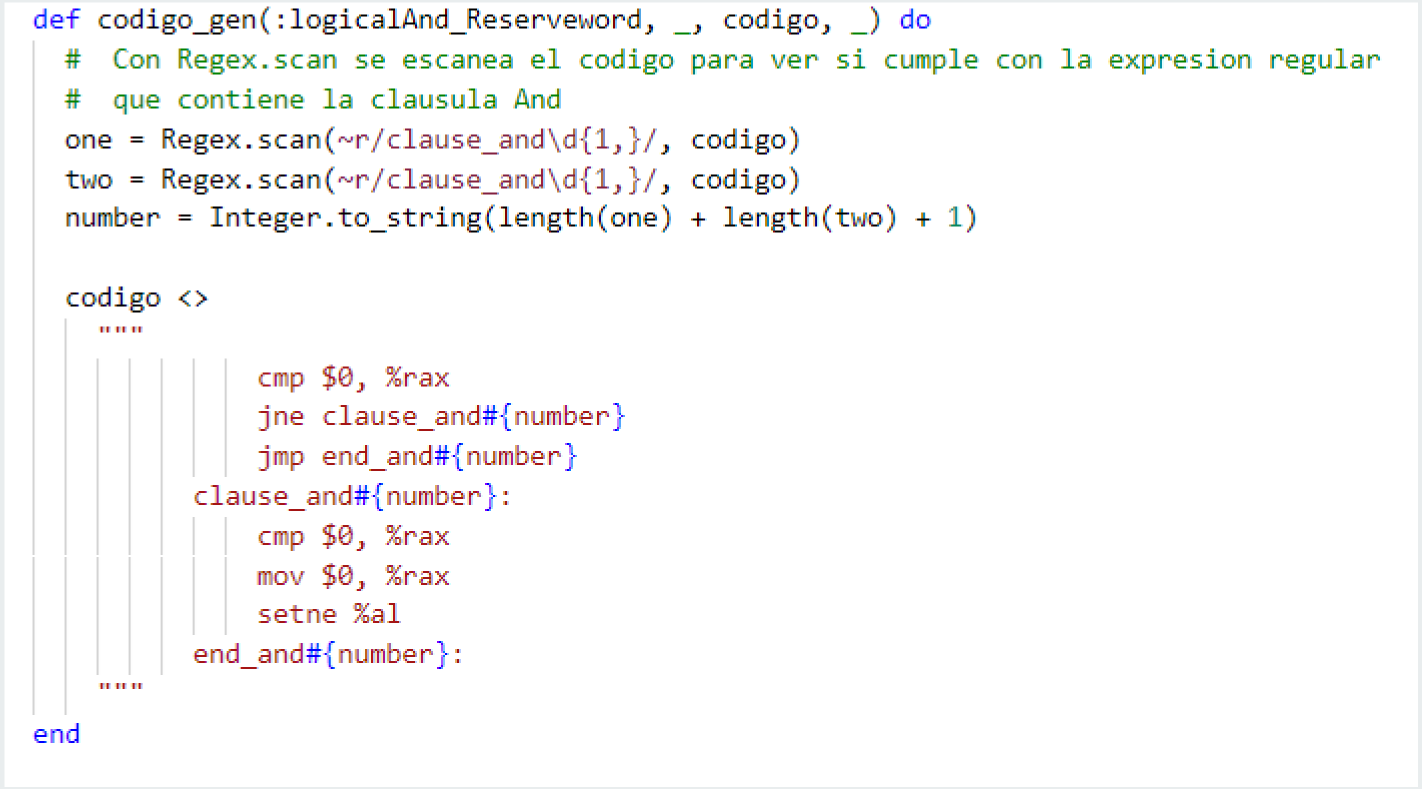
```
def codigo_gen(:constant, value, codigo, post_stack) do
    if "+" in post_stack or "-" in post_stack or "*" in post_stack or "/" in post_stack or ">" in post_stack  or ">=" in post_stack
    or "<=" in post_stack do
        if List.first(post_stack) == "+"
        or List.first(post_stack) == "-"
        or List.first(post_stack) == "*"
        or List.first(post_stack) == "/"
        or List.first(post_stack) == "~"
        or List.first(post_stack) == "!"
        or List.first(post_stack) == ">"
        or List.first(post_stack) == "<"
        or List.first(post_stack) == ">="
        or List.first(post_stack) == "<="do
            codigo <> """
```

Adding binary operators to code generator

```elixir
def codigo_gen(:multiplication_Reserveword, _, codigo, _) do
    codigo <> """
      pop     %rcx
      imul    %ecx, %eax
      push    %rax
    """
end

def codigo_gen(:division_Reserveword, _, codigo, _) do
  codigo <> """
      push    %rax
      pop     %rcx
      pop     %rax
      xor     %edx, %edx
      idivl   %ecx
      push    %rax
  """
end
```

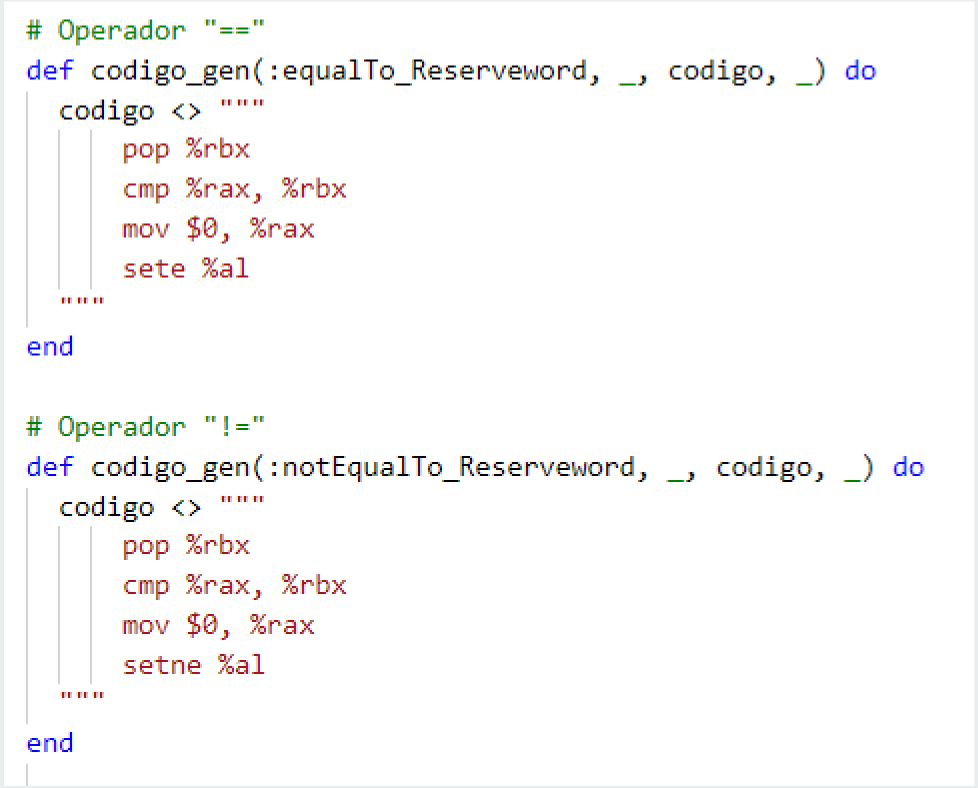
```elixir
def codigo_gen(:min_Reserveword, _, codigo, _) do
    codigo <> """
      pop       %rcx
      sub       %rax, %rcx
      mov       %rcx, %rax
  """
end

def codigo_gen(:add_Reserveword, _, codigo, _) do
    codigo <> """
      pop     %rcx
      addl    %ecx, %eax
      push    %rax
    """
end
```

Changes in Code Generator

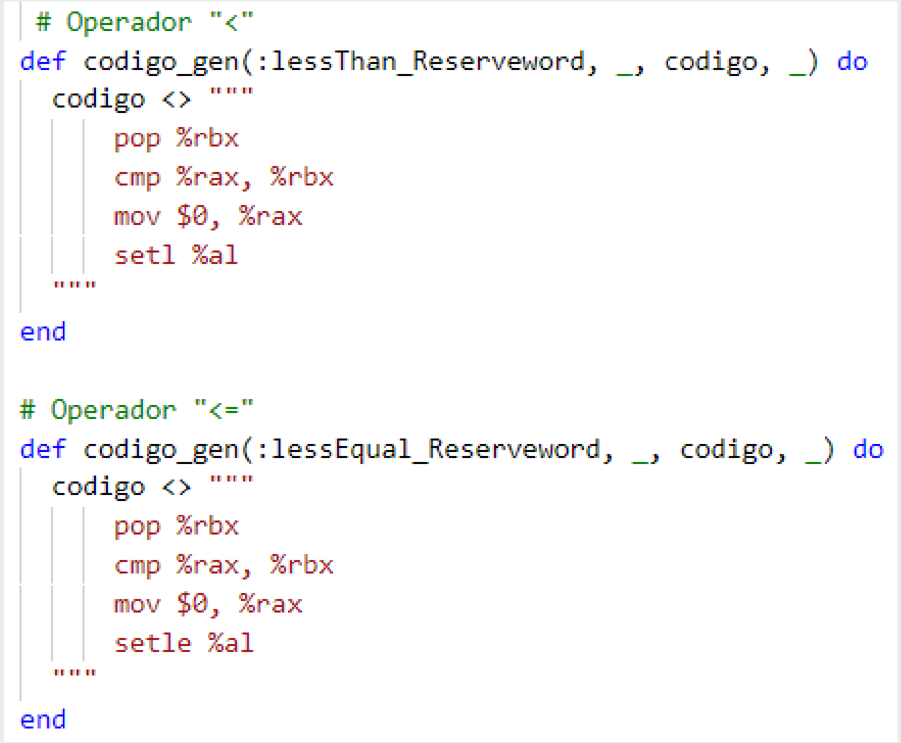


```
def codigo_gen(:logicalAnd_Reserveword, _, codigo, _) do
  #  Con Regex.scan se escanea el codigo para ver si cumple con la expresion regular
  #  que contiene la clausula And
  one = Regex.scan(~r/clause_and\d{1,}/, codigo)
  two = Regex.scan(~r/clause_and\d{1,}/, codigo)
  number = Integer.to_string(length(one) + length(two) + 1)

  codigo <>
    """
              cmp $0, %rax
              jne clause_and#{number}
              jmp end_and#{number}
          clause_and#{number}:
              cmp $0, %rax
              mov $0, %rax
              setne %al
          end_and#{number}:
    """
end
```

Changes in Code Generator

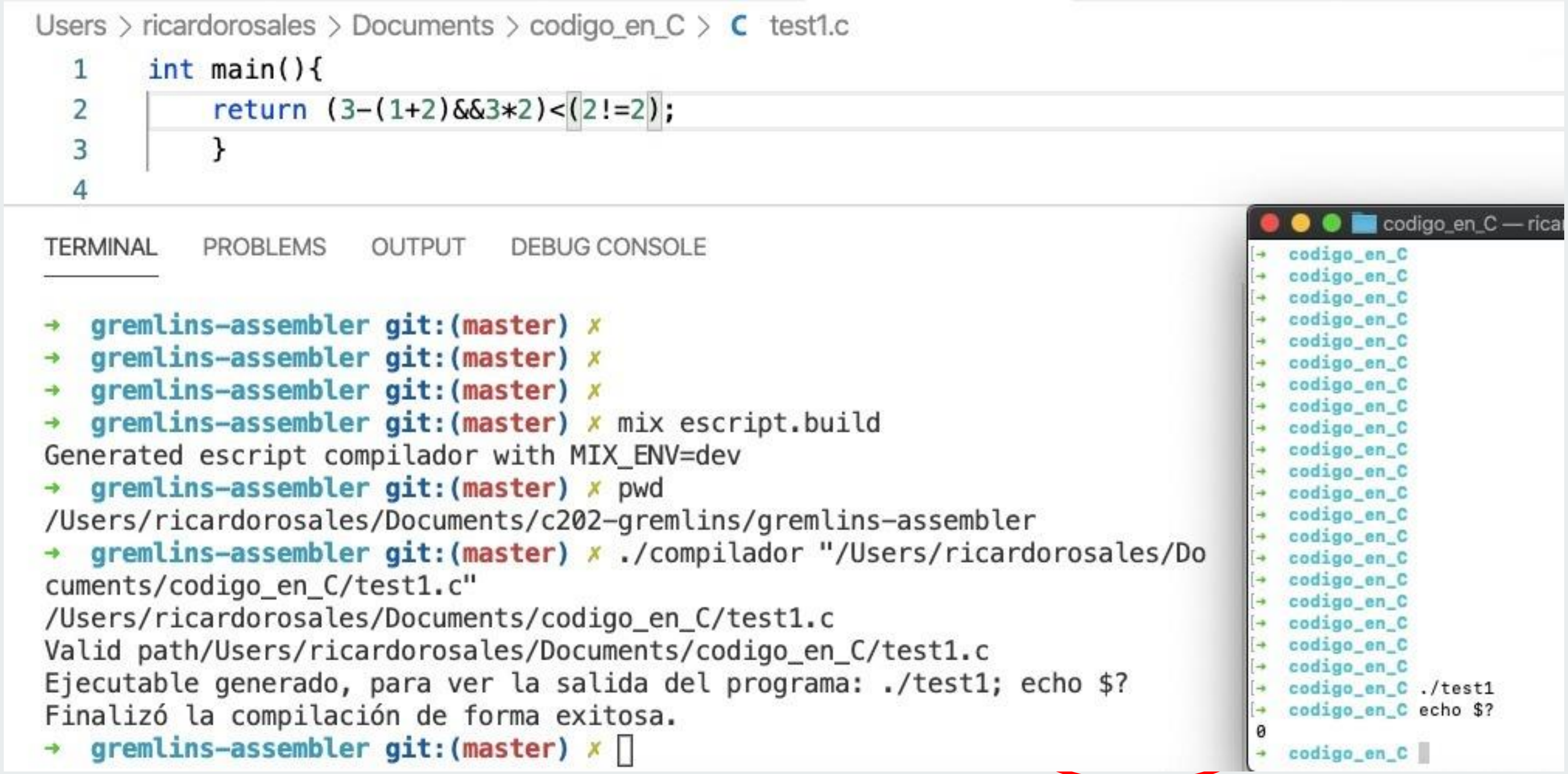


```
# Operador "=="
def codigo_gen(:equalTo_Reserveword, _, codigo, _) do
  codigo <> """
      pop %rbx
      cmp %rax, %rbx
      mov $0, %rax
      sete %al
  """
end

# Operador "!="
def codigo_gen(:notEqualTo_Reserveword, _, codigo, _) do
  codigo <> """
      pop %rbx
      cmp %rax, %rbx
      mov $0, %rax
      setne %al
  """
end
```



```
# Operador "<"
def codigo_gen(:lessThan_Reserveword, _, codigo, _) do
  codigo <> """
      pop %rbx
      cmp %rax, %rbx
      mov $0, %rax
      setl %al
  """
end

# Operador "<="
def codigo_gen(:lessEqual_Reserveword, _, codigo, _) do
  codigo <> """
      pop %rbx
      cmp %rax, %rbx
      mov $0, %rax
      setle %al
  """
end
```

Final test



Users > ricardorosales > Documents > codigo_en_C > C test1.c

```c
int main(){
    return (3-(1+2)&&3*2)<(2!=2);
    }
```

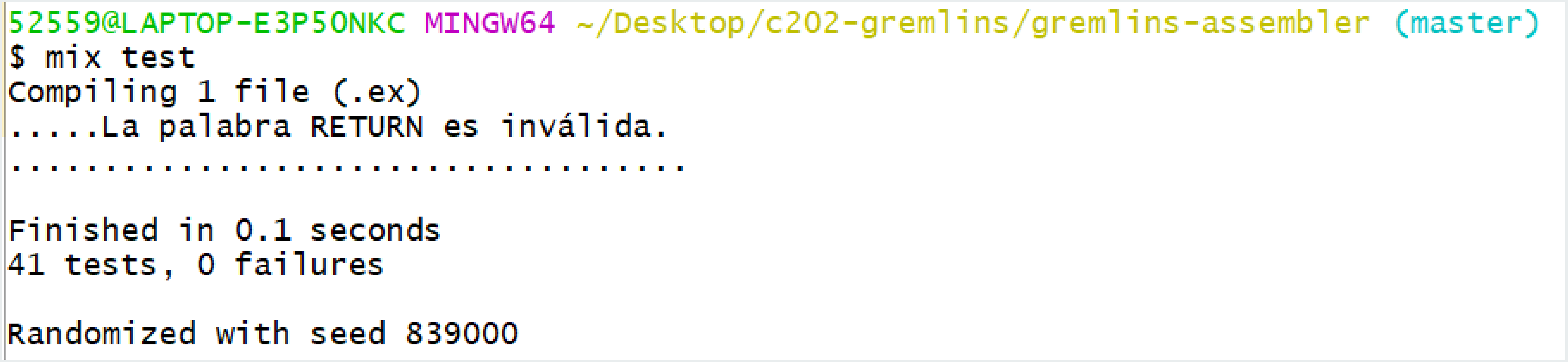
TERMINAL   PROBLEMS   OUTPUT   DEBUG CONSOLE

```
➜  gremlins-assembler git:(master) ✗
➜  gremlins-assembler git:(master) ✗
➜  gremlins-assembler git:(master) ✗
➜  gremlins-assembler git:(master) ✗ mix escript.build
Generated escript compilador with MIX_ENV=dev
➜  gremlins-assembler git:(master) ✗ pwd
/Users/ricardorosales/Documents/c202-gremlins/gremlins-assembler
➜  gremlins-assembler git:(master) ✗ ./compilador "/Users/ricardorosales/Do
cuments/codigo_en_C/test1.c"
/Users/ricardorosales/Documents/codigo_en_C/test1.c
Valid path/Users/ricardorosales/Documents/codigo_en_C/test1.c
Ejecutable generado, para ver la salida del programa: ./test1; echo $?
Finalizó la compilación de forma exitosa.
➜  gremlins-assembler git:(master) ✗ 
```

codigo_en_C — rica

```
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C
[➜  codigo_en_C ./test1
[➜  codigo_en_C echo $?
0
➜  codigo_en_C 
```

Some tests



```
52559@LAPTOP-E3P5ONKC MINGW64 ~/Desktop/c202-gremlins/gremlins-assembler (master)
$ mix test
Compiling 1 file (.ex)
.....La palabra RETURN es inválida.
....................................

Finished in 0.1 seconds
41 tests, 0 failures

Randomized with seed 839000
```

Learned Lessons