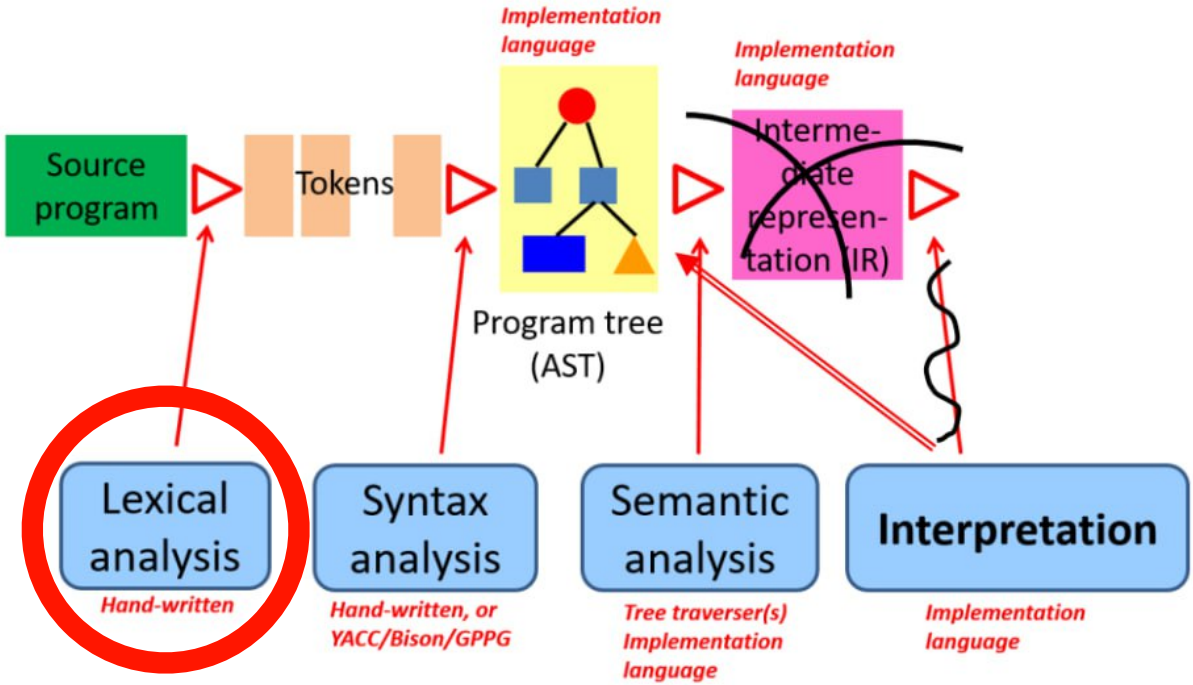
Pechenen team

Lexical analyzer progress

Project F: Functional



Code example of pechenen LISP

GCD

```
(func Fibonacci (n) (
  cond (less n 2)
    (return n)
    (return (plus (Fibonacci (minus n 1)) (Fibonacci (minus n 2)))))
)
```

Lexer output

Fib.out

```
{:ok,
  [
    %Lexer.Token{value: "(", type: :operator, column: 1, line: 1},
    %Lexer.Token{value: :func, type: :atom, column: 2, line: 1},
    %Lexer.Token{value: :Fibonacci, type: :atom, column: 7, line: 1},
    %Lexer.Token{value: "(", type: :operator, column: 17, line: 1},
    %Lexer.Token{value: :n, type: :atom, column: 18, line: 1},
    %Lexer.Token{value: ")", type: :operator, column: 19, line: 1},
    %Lexer.Token{value: "(", type: :operator, column: 21, line: 1},
    %Lexer.Token{value: :cond, type: :atom, column: 2, line: 2},
    %Lexer.Token{value: "(", type: :operator, column: 7, line: 2},
    %Lexer.Token{value: :less, type: :atom, column: 8, line: 2},
    %Lexer.Token{value: :n, type: :atom, column: 13, line: 2},
    ...
  ]
}
```

Code example of pechenen LISP

GCD

```
1 (func Gcd (a b) (
2   cond (equal a b)
3     (return a) (
4     cond (greater a b)
5       (return (Gcd b (minus a b)))
6       (return (Gcd a (minus b a)))
7     )
8   )
9 )
```

Lexer output

GCD.out

```
{:ok,
 [
   %Lexer.Token{value: "(", type: :operator, column: 1, line: 1},
   %Lexer.Token{value: :func, type: :atom, column: 2, line: 1},
   %Lexer.Token{value: :Gcd, type: :atom, column: 7, line: 1},
   %Lexer.Token{value: "(", type: :operator, column: 11, line: 1},
   %Lexer.Token{value: :a, type: :atom, column: 12, line: 1},
   %Lexer.Token{value: :b, type: :atom, column: 14, line: 1},
   %Lexer.Token{value: ")", type: :operator, column: 15, line: 1},
   %Lexer.Token{value: "(", type: :operator, column: 17, line: 1},
   %Lexer.Token{value: :cond, type: :atom, column: 2, line: 2},
   %Lexer.Token{value: "(", type: :operator, column: 7, line: 2},
   %Lexer.Token{value: :equal, type: :atom, column: 8, line: 2},
   %Lexer.Token{value: :a, type: :atom, column: 14, line: 2},
   %Lexer.Token{value: :b, type: :atom, column: 16, line: 2},
   %Lexer.Token{value: ")", type: :operator, column: 17, line: 2},
   %Lexer.Token{value: "(", type: :operator, column: 4, line: 3},
   %Lexer.Token{value: :return, type: :atom, column: 5, line: 3},
   ...
 ]
}
```

Lexer implementation

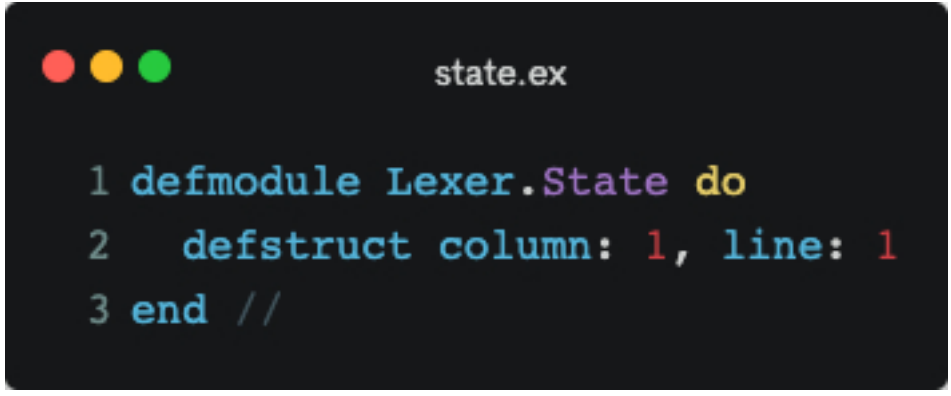
token.ex

```elixir
defmodule Lexer.Token do
  @types ~w(atom liter operator)a

  @type t :: %__MODULE__{
          value: String.t() | boolean() | integer() | float() | atom(),
          type: type()
        }

  @typep type ::
           unquote(
             @types
             |> Enum.map(&inspect/1)
             |> Enum.join(" | ")
             |> Code.string_to_quoted!()
           )
  @enforce_keys [:value, :type, :column, :line]
  defstruct [:value, :type, :column, :line]
end
```

Lexer implementation



Lexer implementation

lexer.ex

```elixir
defp parse_integer(<<value::binary-size(1)>> <> remain, acc)
    when value in ~w(0 1 2 3 4 5 6 7 8 9) do
  parse_integer(remain, acc <> value)
end

defp parse_integer(<<value::binary-size(1)>> <> _remain = code, acc)
    when value in ["." | @delimeters] do
  {acc, code}
end

defp parse_integer("" = remain, acc) do
  {acc, remain}
end

defp parse_integer(remain, acc) do
  {:error, remain, acc}
end
```

Team Contribution

Nikita Pozdniakov

Code for Lexer & Presentation

Maxim Filonov

Code for Lexer & Presentation

Pavel Bakharuev

Presentation & planning

Andrey Sandimirov

Research & collect papers