https://github.com/georgerapeanu/BBU-Computer-Science/tree/master/Semester5/Formal%20Languages%20and%20Compiler%20Design/lab4

This lab works exactly the same as the previous one, but instead of using regexes uses FiniteAutomatons. The finite automaton is implemented in `finite_automaton.rs`. It stores states as strings, and has a variable for the `initial_state`, a hashset for the `final_states` and a hashmap that maps (states(strings), bytes) to the next state.

Also, `cargo run describe automaton` is now available, for seeing information about automaton;

Format of FA.in

```
alphabet                                = "a" | "b" | .... | (any possible character excludi
state                                   = alphabet{alphabet}
final_states_row                        = state{" "state}
transition_row                          = state" "state[" "{alphabet}]
fa.in                                   = state"\n"final_states_row"\n"{transition_row"\n"}
```

A transition row contains the `from` and `to` states as the first two states, and afterwards can contain a list of characters that can transition the `from` state to the `to` state. If it is missing, all the other characters which were not mapped by this point are added as a possible transition from `from` to `to`.

FA.in

```
start
integer_zero integer float string identifier

start integer_zero 0
start signed_integer +-
start integer 123456789
signed_integer integer 123456789
integer integer 0123456789


integer float_dot .
float_dot float 0123456789
float float 0123456789

start inside_string "
inside_string string "
inside_string inside_string

start identifier _abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
identifier identifier _abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

p1.out

```
Program: let x: i32;let y: i32;x = readI32();y = readI32();while y != 0 {let z: i32 = x % y;
PIF.out
let: -
x: 0
:: -
i32: -
;: -
let: -
y: 1
:: -
i32: -
;: -
x: 0
=: -
readI32: -
(: -
): -
;: -
y: 1
=: -
readI32: -
(: -
): -
;: -
while: -
y: 1
!: -
=: -
0: 2
{: -
let: -
z: 3
:: -
i32: -
=: -
x: 0
%: -
y: 1
;: -
x: 0
=: -
y: 1
;: -
y: 1
```

```
=: -
z: 3
;: -
}: -
print: -
(: -
"Gcd is ": 4
,: -
x: 0
): -
;: -
ST.out
x: 0
y: 1
0: 2
"Gcd is ": 4
z: 3
```

p2.out

```
Program: let n: i32 = readI32();let i: i32 = 0;let is_prime: bool = true;i = 2;while i < n {
PIF.out
let: -
n: 0
:: -
i32: -
=: -
readI32: -
(: -
): -
;: -
let: -
i: 1
:: -
i32: -
=: -
0: 2
;: -
let: -
is_prime: 3
:: -
bool: -
=: -
true: -
;: -
i: 1
```

```
=: -
2: 4
;: -
while: -
i: 1
<: -
n: 0
{: -
if: -
n: 0
%: -
i: 1
==: -
0: 2
{: -
is_prime: 3
=: -
false: -
;: -
}: -
i: 1
=: -
i: 1
+: -
1: 5
;: -
}: -
if: -
is_prime: 3
{: -
print: -
(: -
"Number is prime": 6
): -
;: -
}: -
else: -
{: -
print: -
(: -
"Number is not prime": 7
): -
;: -
}: -
ST.out
0: 2
```

```
i: 1
is_prime: 3
"Number is prime": 6
1: 5
"Number is not prime": 7
n: 0
2: 4
```

p3.out

```
Program: let n: u32;let sum: u32 = 0;let i: u32 = 0;n = readU32();while i < n {let val: u32
PIF.out
let: -
n: 0
:: -
u32: -
;: -
let: -
sum: 1
:: -
u32: -
=: -
0: 2
;: -
let: -
i: 3
:: -
u32: -
=: -
0: 2
;: -
n: 0
=: -
readU32: -
(: -
): -
;: -
while: -
i: 3
<: -
n: 0
{: -
let: -
val: 4
:: -
u32: -
```

```
=: -
readU32: -
(: -
): -
;: -
sum: 1
=: -
sum: 1
+: -
val: 4
;: -
i: 3
=: -
i: 3
+: -
1: 5
;: -
}: -
print: -
(: -
"Sum is": 6
,: -
sum: 1
): -
;: -
ST.out
0: 2
i: 3
"Sum is": 6
sum: 1
1: 5
n: 0
val: 4
```

p1err.out

```
Program: let 0a: i32; let x = 12_34;
thread 'main' panicked at src/main.rs:81:7:
Lexical error for token number 2
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```