COMPILER PROJECT I 2022 - Documentation

Tokens & Regular Expressions

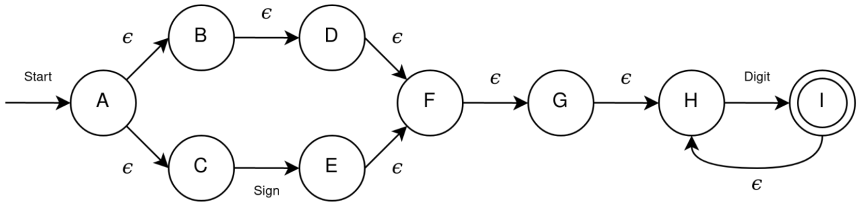
(arithmetic / assignment / comparison / symbols)

Signed integer:

Σ = {0, 1, 2, …, 9, -}

Digit = 0|1|2|…|9

r = (-|ϵ)(Digit)$^+$



T0 = ϵ-closure(A) = {}
T1 = ϵ-closure(δ(T0, Digit)) = ϵ-closure(I) = {I, H}
T2 = ϵ-closure(δ(T0, Sign)) = ϵ-closure(E) = {E, F, G, H}
ϵ-closure(δ(T1, Digit)) = T1
ϵ-closure(δ(T1, Sign)) = Ø
ϵ-closure(δ(T2, Digit)) = T1
ϵ-closure(δ(T2, Sign)) = Ø

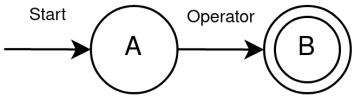
	Letter	Digit
T0	T1	T2
T1	T1	Ø
T2	T1	Ø

Operators:

Σ = {+, -, *, /, =, <, >, ==, !=, <=, >=, ;, {, }, (,), ,}

Operator = +|-|*|/|=|<|>|==|!=|<=|>=|;|{|}|(|)|,

r = Operator



	Operator
T0	T1
T1	Ø

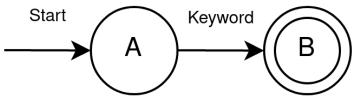
T0 = ϵ-closure(A) = {A}
T1 = ϵ-closure(δ(T0, Operator)) = {B}
ϵ-closure(δ(T1, Operator)) = Ø

Keywords:

Σ = {if, IF, else, ELSE, while, WHILE, return, RETURN, int, INT, char, CHAR}

Keyword = if|IF|else|ELSE|while|WHILE|return|RETURN|int|INT|char|CHAR

r = Keyword



T0 = ϵ-closure(A) = {A}
T1 = ϵ-closure(δ(T0, Keyword)) = {B}
ϵ-closure(δ(T1, Keyword)) = Ø

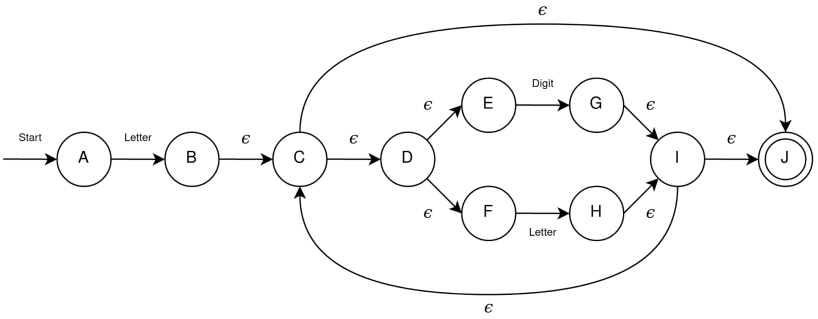
	Keyword
T0	T1
T1	Ø

Identifiers:

Σ = {A, B, C, …, Z, a, b, c, …, z, 0, 1, 2, …, 9}

Letter = A|B|C|…|Z|a|b|c|…|z
Digit = 0|1|2|…|9

r = Letter(Letter|Digit)*



T0 = ϵ-closure(A) = {A}
T1 = ϵ-closure(δ(T0, Letter)) = ϵ-closure(B) = {B, C, D, E, F, J}, ϵ-closure(δ(T0, Digit)) = Ø
T2 = ϵ-closure(δ(T1, Letter)) = ϵ-closure(H) = {D, E, F, H, I, J}
T3 = ϵ-closure(δ(T1, Digit)) = ϵ-closure(G) = {D, E, F, G, I, J}
ϵ-closure(δ(T2, Letter)) = {D, E, F, H, I, J} = T2
ϵ-closure(δ(T2, Digit)) = {D, E, F, G, I, J} = T3
ϵ-closure(δ(T3, Letter)) = {D, E, F, H, I, J} = T2
ϵ-closure(δ(T3, Digit)) = {D, E, F, G, I, J} = T3

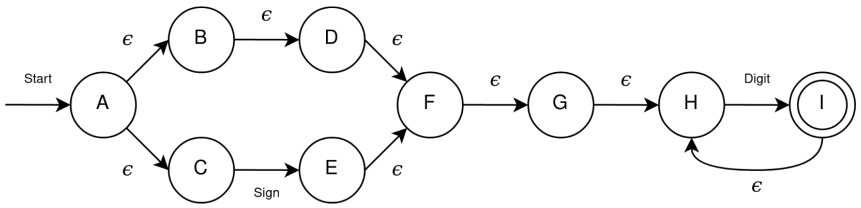
	Letter	Digit
T0	T1	Ø
T1	T2	T3
T2	T2	T3
T3	T2	T3

Signed integer:

Σ = {0, 1, 2, …, 9, -}

Digit = 0|1|2|…|9

r = (-|ϵ)(Digit)$^+$



T0 = ϵ-closure(A) = {}
T1 = ϵ-closure(δ(T0, Digit)) = ϵ-closure(I) = {I, H}
T2 = ϵ-closure(δ(T0, Sign)) = ϵ-closure(E) = {E, F, G, H}
ϵ-closure(δ(T1, Digit)) = T1
ϵ-closure(δ(T1, Sign)) = Ø
ϵ-closure(δ(T2, Digit)) = T1
ϵ-closure(δ(T2, Sign)) = Ø

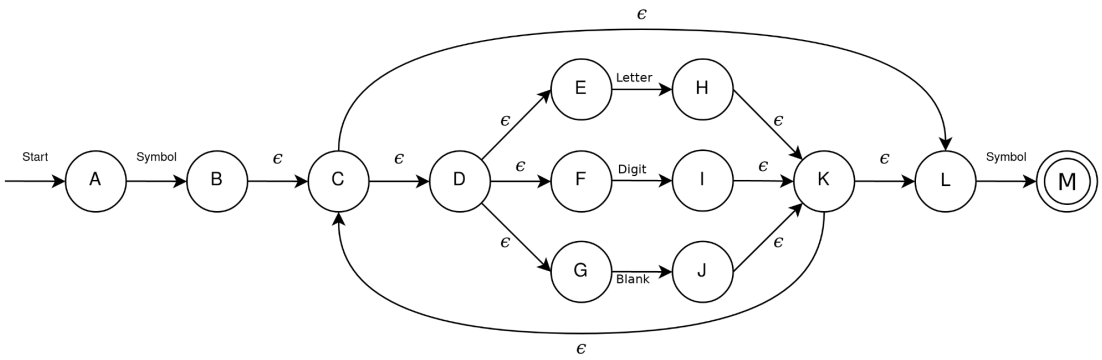
	Letter	Digit
T0	T1	T2
T1	T1	Ø
T2	T1	Ø

Literal string:

Σ = {A, B, C, …, Z, a, b, c, …, z, 0, 1, 2, …, 9, , “}

Letter = A|B|C|…|Z|a|b|c|…|z
Digit = 0|1|2|…|9

r = “(Letter|Digit|)*"



T0 = ϵ-closure(A) = {A}
T1 = ϵ-closure(δ(T0, Symbol)) = ϵ-closure(B) = {B, C, D, E, F, L}
ϵ-closure(δ(T0, Symbol)) = Ø
ϵ-closure(δ(T0, Symbol)) = Ø
ϵ-closure(δ(T0, Symbol)) = Ø
T2 = ϵ-closure(δ(T1, Letter)) = ϵ-closure(H) = {H, K, L, C, D, E, F, G}
T3 = ϵ-closure(δ(T1, Digit)) = ϵ-closure(I) = {I, K, L, C, D, E, F, G}
T4 = ϵ-closure(δ(T1, Blank)) = ϵ-closure(J) = {J, K, L, C, D, E, F, G}
T5 = ϵ-closure(δ(T1, Symbol)) = ϵ-closure(M) = {M}
ϵ-closure(δ(T2, Letter)) = T2
ϵ-closure(δ(T2, Digit)) = T3
ϵ-closure(δ(T2, Blank)) = T4
ϵ-closure(δ(T2, Symbol)) = T5
ϵ-closure(δ(T3, Letter)) = T2
ϵ-closure(δ(T3, Digit)) = T3
ϵ-closure(δ(T3, Blank)) = T4
ϵ-closure(δ(T3, Symbol)) = T5
ϵ-closure(δ(T4, Letter)) = T2
ϵ-closure(δ(T4, Digit)) = T3
ϵ-closure(δ(T4, Blank)) = T4
ϵ-closure(δ(T4, Symbol)) = T5
ϵ-closure(δ(T5, Letter)) = Ø
ϵ-closure(δ(T5, Digit)) = Ø
ϵ-closure(δ(T5, Blank)) = Ø
ϵ-closure(δ(T5, Symbol)) = Ø

	Letter	Digit	Blank	Symbol
T0	Ø	Ø	Ø	T1
T1	T2	T3	T4	T5
T2	T2	T3	T4	T5
T3	T2	T3	T4	T5
T4	T2	T3	T4	T5
T5	Ø	Ø	Ø	Ø

Implementation details

We have made a diagram (projectRoot/diagram.png) showing the overall procedures of our lexical analysis. This will help in understanding the flow of the program and how we proceed to recognize tokens.

In the file 'lexical_analyser.cpp', contains the loop in order to check for tokens among the string from the input file given to the program. Instead of calling every function (string_analyser, integer_analyser, assignement_analyser, etc…) that were made to check the type of the token, we decided to create an array of pointer functions for better visibility of the code.

The pointer function was implemented like that:

```
bool (*func_ptrs[])(
    const std::string &,
    std::size_t &,
    Scanner &
) = {
    vtype_analyzer,
    integer_analyzer,
    string_analyzer,
    identifier_analyzer,
    keyword_analyzer,
    arithmetic_analyzer,
    comparison_analyzer,
    assignment_analyzer,
    semi_analyzer,
    brace_analyzer,
    paren_analyzer,
    comma_analyzer
};
```

As you can see, this array will only contains function that follow this specific function signature : **bool** funcname**(const std::string &, std::size_t &, Scanner &).**

We will iterate over this array while sending to each of these functions: our input string, the current index, and our Scanner structure containing a list of our tokens found during the lexical analysis. The loop will continue until one of the analyzers returns a true boolean, which means that a new token has been found and pushed to our list of tokens, or until we reach the end of the array meaning it will result in throwing an error because no token has been found.