# Lexical Analyzer

In this project, you will build a small compiler for mini-C language. In the first phase of this project, you are required to build a program that tokenizes a stream of characters and outputs tokens in the order they appeared in the stream of characters.

- You can use regex library in this phase but do not use JavaCC.
- **The input** of this program is a stream of characters read from a file.
- **Output** is list of Tokens each token will be from class Token that you will build. This class has TYPE, which defines type of each token, and VALUE, which is the string it matched from the stream of characters.
- **Output** should be saved in file too.
- Error message should apperate if your program can't match with any Token. Ex 2x=5; -> no ID start with number.
- Deadline: 22/3/2019 at 11:55 PM @Acadox
- Number of members per group is 4 and no exceptional cases.
- Your submission will be on Acadox as zip or rar file.
- Name of the submitted file should follow P1_Lab#_ID1_ID2_ID3_ID4.rar

Lab1 if you are from CS_IS_3 or CS_IS_4
Lab2 => CS_IS_1 or CS_IS_2
Lab3 => CS_IT or CS_DS
Ex: lab1_2014000_20140001_20140002_20140003.rar

## List of Tokens:

- Write the suitable regular expressions for these tokens.
- Use these token names/ labels as a unified name between us when you refer to the token name. For example, <AUTO> or <COMMENT1> etc...

| Token name/ Label | Tiny C Token | Regular Expression |
|---|---|---|
| AUTO | auto | |
| NEW | new | |
| EOF | 0 | |
| TRUE | true | |
| FALSE | false | |
| BREAK | break | |
| BOOL | bool | |
| CASE | case | |
| CHAR | char | |
| CONST | const | |
| CONTINUE | continue | |
| DEFAULT | default | |
| DO | do | |
| DOUBLE | double | |

| | | |
|---|---|---|
| ELSE | else | |
| ENUM | enum | |
| EXTERN | extern | |
| FLOAT | float | |
| FOR | for | |
| GOTO | goto | |
| IF | if | |
| INT | int | |
| LONG | long | |
| REGISTER | register | |
| RETURN | return | |
| SHORT | short | |
| SIGNED | signed | |
| SIZEOF | sizeof | |
| STATIC | static | |
| STRUCT | struct | |
| SWITCH | switch | |
| TYPEDEF | typedef | |
| UNION | union | |
| UNSIGNED | unsigned | |
| VOID | void | |
| VOLATILE | volatile | |
| WHILE | while | |
| *ID* | *Any valid identifier*<br>*(Note: ID can't start with num)* | |
| *INTEGRAL_LITERAL* | 0, 1, 1218, 12482 | |
| *FLOAT_LITERAL* | 30486.184 ,1203.03 ,0.0 | |
| *STRING_LITERAL* | "bla bla" | |
| *CHAR_LITERAL* | 'a' , 'z' | |
| *LEFT_CURLY_B* | } | |
| *RIGHT_CURLY_B* | { | |
| *LEFT_SQUARE_B* | ] | |
| *RIGHT_SQUARE_B* | [ | |
| *LEFT_ROUND_B* | ) | |
| *RIGHT_ROUND_B* | ( | |
| *COMMA* | , | |
| *SEMICOLON* | ; | |
| *DOT* | . | |
| *NOT* | ! | |
| *ASSIGN_OPERATOR* | = | |
| *PREPROCESSOR* | # | |
| *BACKWARD_SLASH* | \ | |
| *MINUS* | - | |
| *PLUS* | + | |
| *ASTERICK* | * | |
| *DIVIDE* | / | |
| *MOD* | % | |
| *LESSTHAN* | > | |

| | | |
|---|---|---|
| *GREATERTHAN* | < | |
| *LESS_EQ* | => | |
| *GREAT_EQ* | =< | |
| *EQUAL* | == | |
| *NOT_EQUAL* | != | |
| *AND* | && | |
| *OR* | \|\| | |
| *BITWISE_AND* | & | |
| *BITWISE_OR* | \| | |
| *BITWISE_XOR* | ^ | |
| *LEFT_SHIFT* | >> | |
| *RIGHT_SHIFT* | << | |
| *BITWISE_NOT* | ~ | |
| *MULTI_COMMENT* | /* blabla */ | |
| *SINGLE_COMMENT* | // | |

## Examples:

### Example 1:

Input:

```
int intvalue = 10+5;
```

Output:

<INT> : int
<ID> : intvalue
<***ASSIGN_OPERATOR***> : =
<INTEGER_LITERAL> : 10
<PLUS> : +
<INTEGER_LITERAL> : 5
<SEMICOLON> : ;

Note:

- No spaces between "10+5;"
- Order of tokens is important.

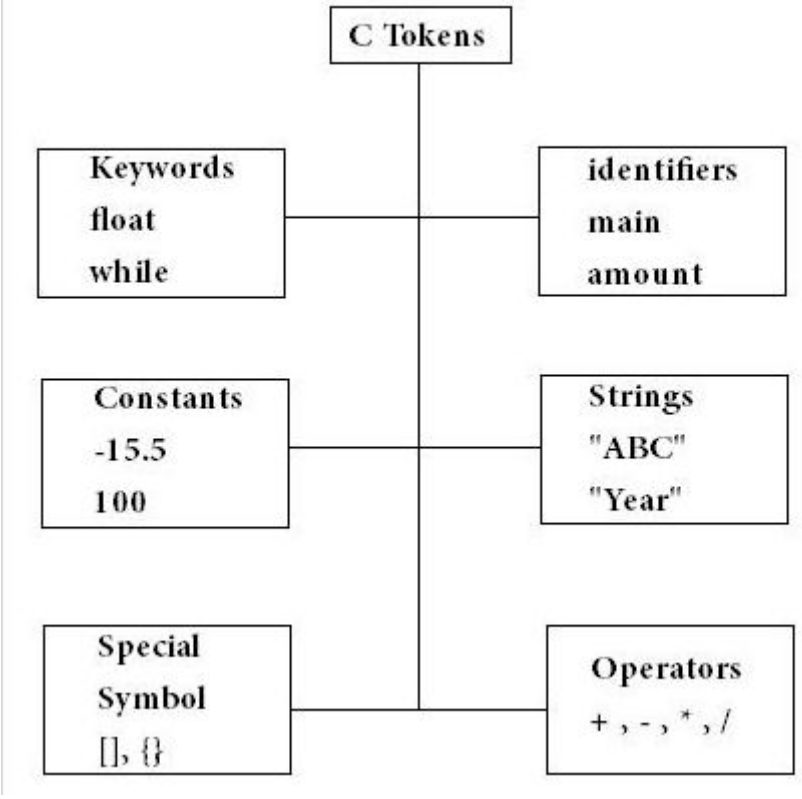
### Example 2:

Input:

```
bool isPowerOfTwo(int x)
{
    // First x in the below expression is
    // for  the case when x is 0
    return x && (!(x & (x - 1)));
}
```

Output:

<BOOL> : bool
<ID> : isPowerOfTwo
< LEFT_ROUND_B> : (
<INT> : int
<ID> : x
<RIGHT_ROUND_B> : )
<LEFT_CURLY_B> : {
<SINGLE_COMMENT> : // First x in the below expression is
<SINGLE_COMMENT> : // for  the case when x is 0
<RETURN> : return
<ID>: x
<AND>: &&
< LEFT_ROUND_B> : (
<NOT> !
< LEFT_ROUND_B> : (
<ID>: x
< BITWISE_AND>: &
< LEFT_ROUND_B> : (
<ID>: x
<MINUS>: -
<INTEGER_LITERAL>: 1
<RIGHT_ROUND_B> : )
<RIGHT_ROUND_B> : )
<RIGHT_ROUND_B> : )
<SEMICOLON>: ;
<RIGHT_CURLY_B> : }


C Tokens
Keywords
float
while
identifiers
main
amount
Constants
-15.5
100
Strings
"ABC"
"Year"
Special
Symbol
[], {}
Operators
+ , - , * , /