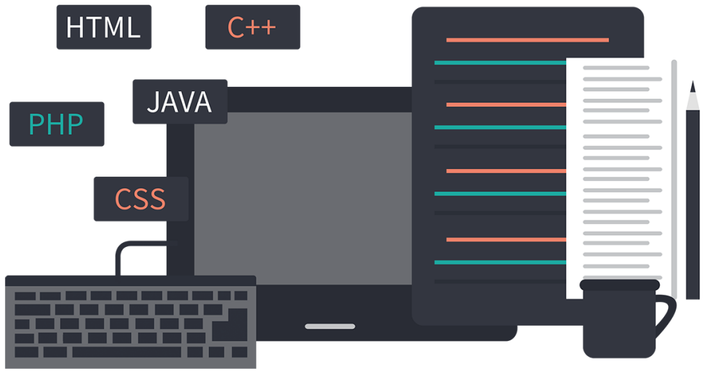


COMPILER DESIGN PROJECT

STUDENTS NAMES:

Bahsayer Bajaber |438018481
Sarah Mansour | 438018452
Lubaba Saeed | 438018566

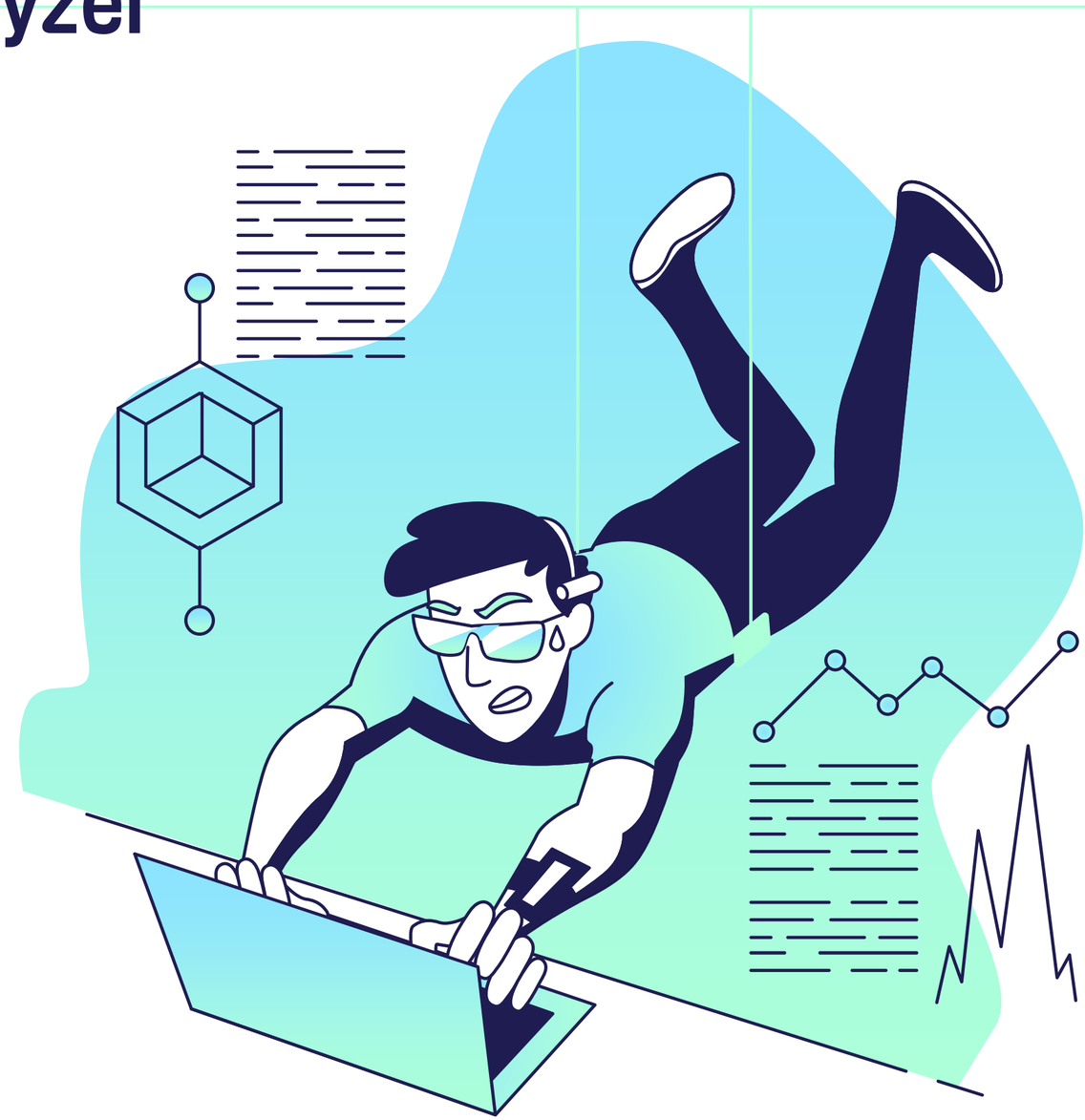
Tasks Dividing

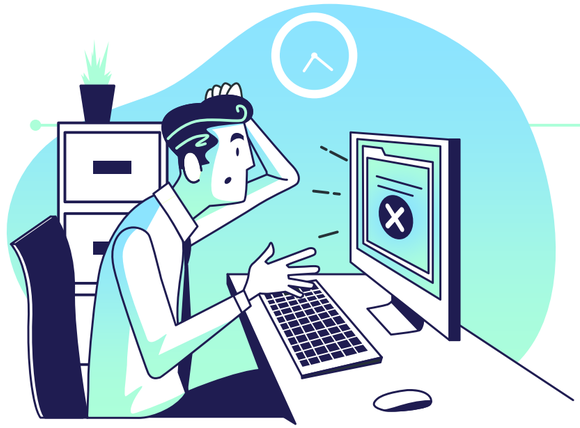
Tasks	Bashayer	Sarah	Lubaba
Part1 task1: collect info about lex	.	.	.
Write info of lex		.	
Part1 task2: Learn how to write	.	.	.
Write lex code			.
Part1 task3 Learn how to write	.	.	.
Write symbol table code	.	.	
Part 2 task1: collect info about yacc	.	.	.
Write info of yacc	.		.

Part1 – Task 1: Lexical Analyzer

What is the Lex?

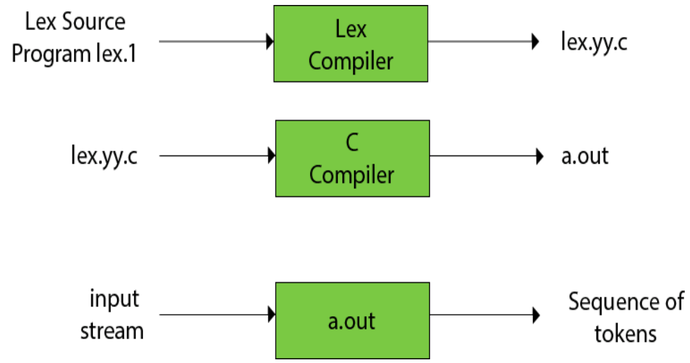
Lex is a tool used in the first phase of the compiler Lexical Analyzer phase recognizes regular expressions and generates tokens. The Lex code recognizes expressions in an input stream and partitions the input stream into strings matching the expressions. A string of entries it reads, recognizes and combined with the source code to generate token.





How it works? Is there a specific syntax for the Lex code?

- Firstly lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
- Finally C compiler runs the lex.yy.c program and produces an object program a.out.
- a.out is lexical analyzer that transforms an input stream into a sequence of tokens.



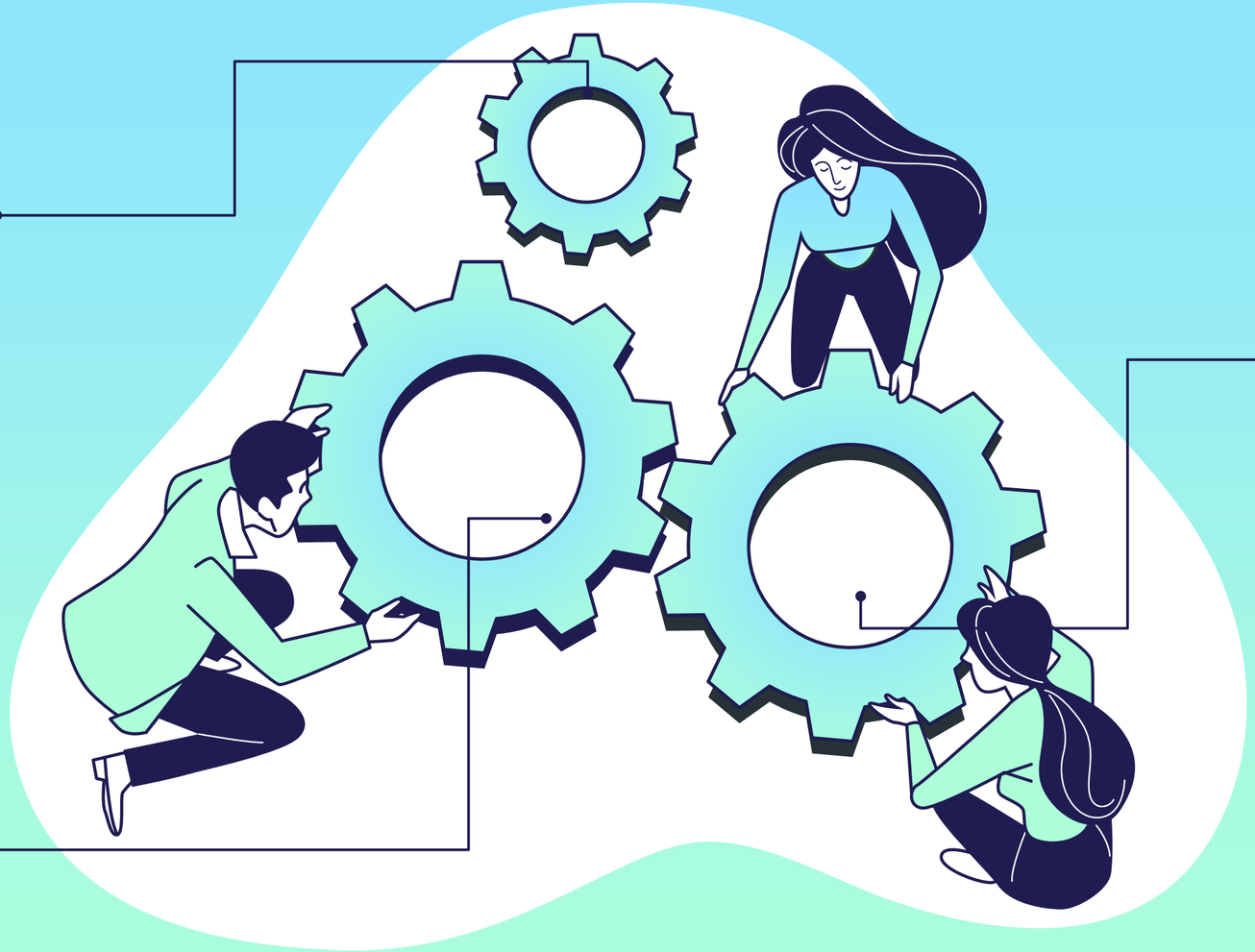
Lex file format :

1. { definitions }
2. %%
3. { rules }
4. %%
5. { user subroutines }

- A Lex program is separated into three sections by %% delimiters. The formal of Lex source is as follows:
- **Definitions** include declarations of constant, variable and regular definitions.
- **Rules** define the statement of form p1 {action1} p2 {action2}....pn {action}.
- Where **pi** describes the regular expression and **action1** describes the actions what action the lexical analyzer should take when pattern pi matches a lexeme.
- **User subroutines** are auxiliary procedures needed by the actions. The subroutine can be loaded with the lexical analyzer and compiled separately.

Can we use IDE to write a Lex code?

Visual studio



ANTLR

Flex windows

PART 1- TASK 2 :
LEX CODE

```
%{
#include<stdio.h>
%}
WS ([ ]|[\t]|[\n])+
letter [a-zA-Z]
digit [0-9]
Identifier {letter}({letter}|{digit}|[_])*
char [a-zA-Z]
datatype "char"|"int"|"double"|"string"|"float"
STR ["]({char}*[ ]*)*["]
SYM "+"|"-"|"*"|"="|"<"|"<="|">"|">="|"<>"|","|":"|";"|"."|":="|"("|")"|"["|"]"|".."
Comments [{]([ ]*{char}*{digit}*{SYM}*)*[}]
Keyword "if"|"else"|"while"|"do"|"switch"|"case"|"and"|"begin"|"forward"|"div"|"end"|"for"|"function"|"array"|"mod"|"not"|of"|"or"|"procedure"|"program"|"record"|"then"|"to"|"type"|var"|"while"
%%
{WS} {printf(" ");}
{digit}+ {printf("INT");}
{digit}+[.]{digit}+ {printf("double");}
{datatype} {printf("datatype");}
[']{char}['] {printf("char");}
{SYM} {printf("SYM");}
{STR} {printf("STR");}
{Keyword} {printf("Keyword");}
{Identifier} {printf("Identifier");}
{Comments}   ;
({digit}+|[_])+{Identifier} {printf("invalid_Identifier");}
{digit}*[_]+{digit}*  {printf("invalid");}
.  {printf("ERORR!");}
%%
int yywrap()
{
return 1;
}
main()
{
printf("Enter a string of data\n");
yylex();
}
```

Output

```
Enter a string of data
char c = 'b' ;
datatype Identifier SYM char SYM
int i = 90 ;
 datatype Identifier SYM INT SYM
double d = 6.7
 datatype Identifier SYM double
while ( )
 Keyword SYM SYM
if ( ) {if statment}
 Keyword SYM SYM
end
 Keyword
```

PART 1- TASK 3 :
SYMBOL TABLE CODE

Symbol Table is an important data structure created and maintained by the compiler in order to keep track of semantics of variable i.e. it stores information about scope and binding information about names, information about instances of various entities such as variable and function names, classes, objects, etc.

•It is built in lexical and syntax analysis phases.

•The information is collected by the analysis phases of compiler and is used by synthesis phases of compiler to generate code.

•It is used by compiler to achieve compile time efficiency.

•It is used by various phases of compiler

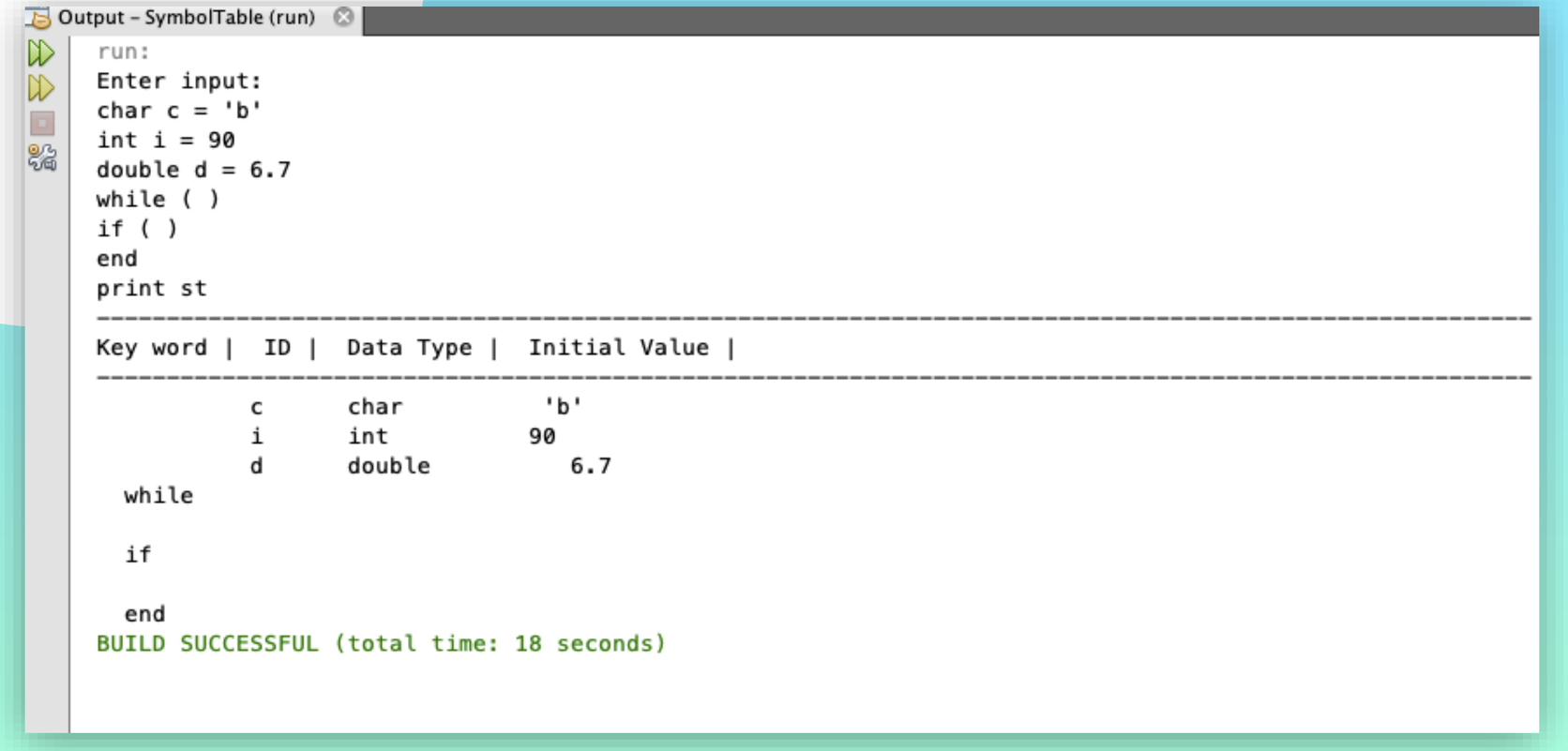
```
package symboltable;
import java.util.Scanner;

public class SymbolTable {
    public static void main(String[] args) {
        String dataType[]= {"int","string","double","char","float"};
        String keyWords [] = {"for","function","if","var","while","else","do","switch","case","and",
            "begin","forward","div","end","array","mod","not","of","or","procedure","program","record","then","to","type"};
        int n;
        Scanner s = new Scanner(System.in);
        String table [][]= new String[][]{{"Key word","ID","Data Type","Initial Value"}};
        System.out.println("Enter input: ");
        String input ="";
        while (s.hasNext()){
        input+= s.nextLine() + " ";
        if (input.contains("print st")){
        break;
        }
        }
        String arr[]= input.split(" ");
        System.out.println("-----------------------------------------------------------------------------------------------------------------");
        for (int i =0; i<4;i++){
            System.out.print(table[0][i]+ " |  ");
        }
        System.out.println("");
        System.out.println("-----------------------------------------------------------------------------------------------------------------");
        int pos =1;
        for (int i =0; i<arr.length;i++){
            for (int k=0;k<keyWords.length;k++){
            if (arr[i].equals(keyWords[k])){
                 switch (arr[i]){
                     case "else" :
                     case "do" :
                     case "case":
                     case "and":
                     case "begin":
```

SYMBOL TABLE CODE

```
        case "begin":
        case "forward":
        case "div":
        case "end":
        case "array":
        case "mod":
        case "not":
        case "of":
        case "or":
        case "procedure":
        case "program":
        case "record":
        case "then":
        case "to":
        case "type":
        case "var" :
        System.out.print("  "+arr[i]+"      ");
            System.out.println(" ");
    }
}}
for (int j =0; j<dataType.length;j++){

    if (arr[0].equals(dataType[j])){

        switch (arr[i]){
            case "=":{
            pos=i;
                System.out.print("           "+arr[pos-1]+"");// print id
                 System.out.print("      "+arr[pos-2]+"          "); //print datatype

                System.out.println(arr[pos+1]+"");// initial value
        }
            break;
            case "(":{
            pos = i;
            n=i+1;
```

```
                    System.out.println(arr[pos+1]+"");// initial value
                }
                    break;
                    case "(":{
                    pos = i;
                    n=i+1;
                        System.out.print("  "+arr[pos-1]+"      "); // print keyword
                        if(arr[pos+1].equals(")")){
                            System.out.println("             ");
                        }
                        }
                        System.out.println("");
                    break;
                    default:

                        break;
                }
            }
        }
    }


}
}
```

Output


Output – SymbolTable (run)
run:
Enter input:
char c = 'b'
int i = 90
double d = 6.7
while ( )
if ( )
end
print st
Key word | ID | Data Type | Initial Value |
c char 'b'
i int 90
d double 6.7
while
if
end
BUILD SUCCESSFUL (total time: 18 seconds)

PART 2 – TASK 1 Syntax Analyzer

What is the YACC?

YACC is an abbreviation for (yet another compiler-compiler) was originally designed for being complemented by Lex.
it is a tool that is used in the second phase of compiler the Syntax Analyzer to generate the Parser after lex converts the source program into tokens take it as input and produces parser as output.
So it is used to give sturcure to tokens and correlate it.

How it works? How it can work with the lex? What is the structure if YACC program?

A YACC program consists of three sections: Declarations, Rules and Auxiliary functions. (Note the similarity with the structure of LEX programs).

1.DECLARATIONS
2.%%
3. RULES
4.%%
5.AUXILIARY FUNCTIONS

Use the system tool YACC to check the validity of an input language. yacc generates parsers, programs that analyze input to insure that it is syntactically correct.
lex and yacc often work well together for developing compilers.
a program uses the lex-generated scanner by repeatedly calling the function **yylex()**. This name is convenient because a yacc-generated parser calls its lexical analyzer with this name.
To use lex to create the lexical analyzer for a compiler, end each lex action with the statement return token, where token is a defined term with an integer value.

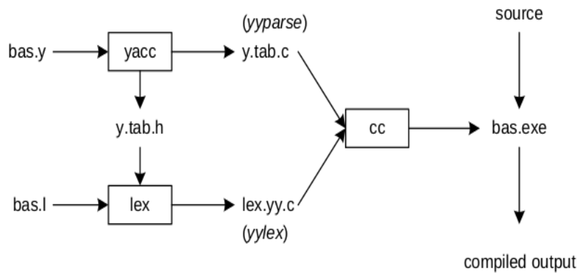


Figure 2: Building a Compiler with Lex/Yacc

Challenges

We tried saving the entries in the lex code in an external file and using this file as an input in the Symbol Table code but it does not success

We tried including the Symbol Table code with the lex code but it does not success

Compile and run the Lex and yacc programs onto many platforms like: windows, ubuntu

We usually understand any new topic from videos and that was a big challenge because the videos resources that teach lex and yacc was limited

REFERENCES

- http://dinosaur.compilertools.net/lex/index.htmlSafe concept illustration
- https://www.javatpoint.com/lexLogin concept illustration
- https://stackoverflow.com/questions/1150301/flex-bison-ide
- https://www.geeksforgeeks.org/symbol-table-compiler/
- https://www.youtube.com/watch?v=F5E8MV80QKU&feature=youtu.be
- https://silcnitc.github.io/yacc.html
- https://www.geeksforgeeks.org/introduction-to-yacc/
- https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dgo/index.html
- https://youtu.be/AI-jwky_mqM
- https://www.cs.fsu.edu/~engelen/cscan.html
- https://docs.oracle.com/cd/E19504-01/802-5880/6i9k05dgk/index.html#lex-36741
- http://osr507doc.sco.com/en/tools/Lex_yacc.html
- https://www.ibm.com/docs/en/zos/2.4.0?topic=works-yyparse-yylex
- https://www.thecrazyprogrammer.com/2017/02/lexical-analyzer-in-c.html
- https://www.csd.uwo.ca/~mmorenom/CS447/Lectures/Lexical.html/node11.html
- https://www.youtube.com/watch?v=g3Gka9PDrgI
- https://www.youtube.com/watch?v=54bo1qaHAfk