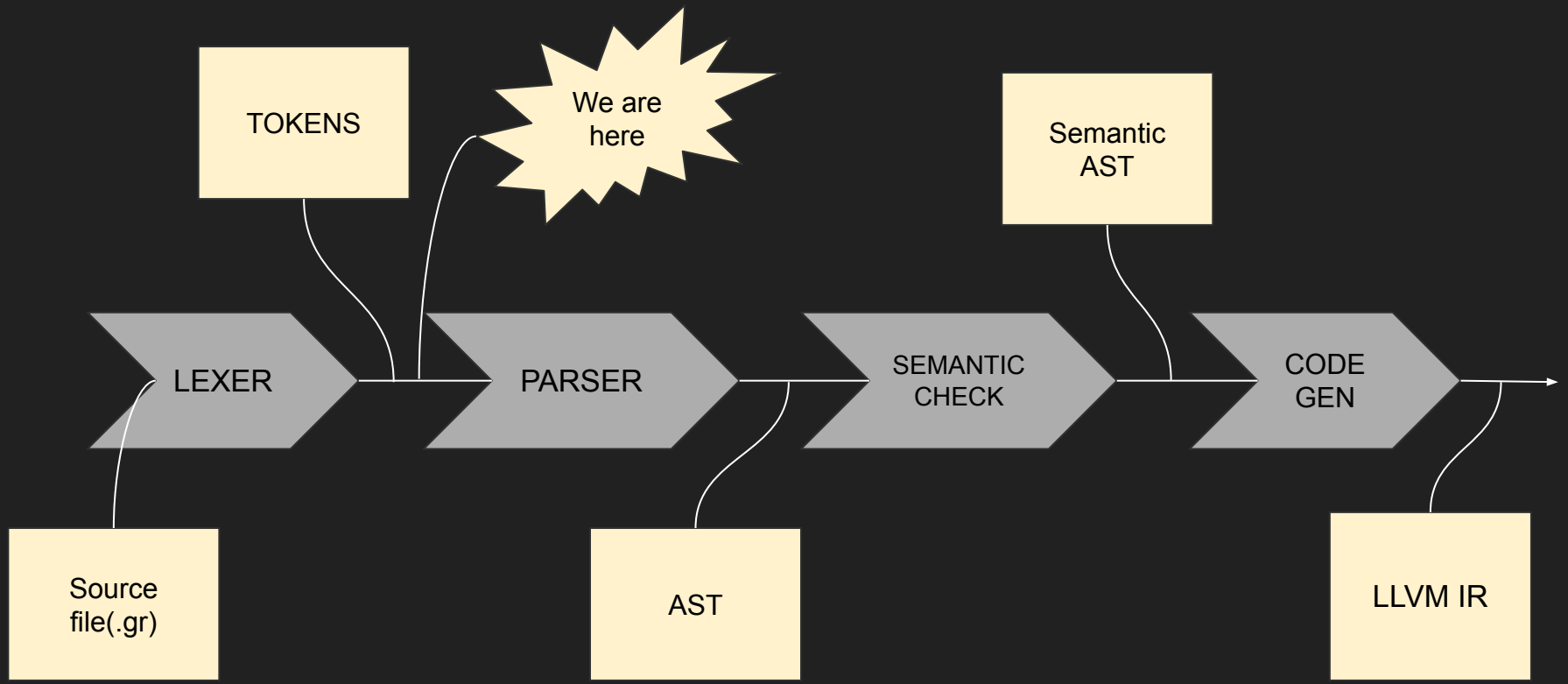
GrAlgo

Group 5

Sahil Chandra- CS20BTECH11033
P. Ganesh Nikhil Madhav- CS20BTECH11036
Gorantla Pranav Sai- CS20BTECH11018
Suraj Telugu- CS20BTECH11050
Umesh Kalvakuntla- CS20BTECH11024
Vanga Aravind Shounik- CS20BTECH11055
Adepu Vasisht- CS20BTECH11002

How does a compiler work?



Lexical Analysis

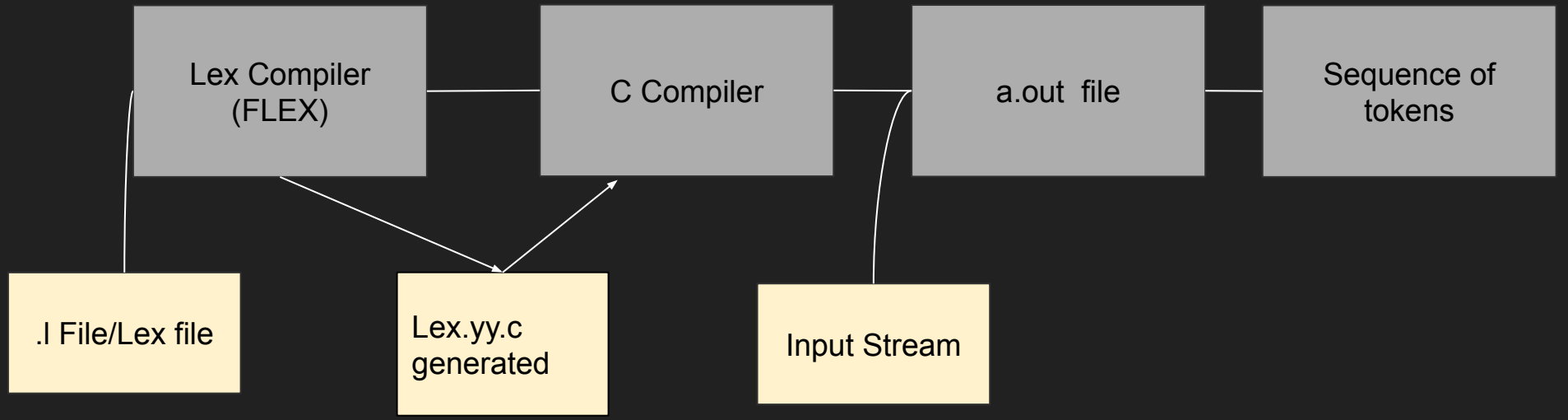
- Lexical Analysis is the first phase of compiler as seen from the flowchart above, it takes a modified source code and converts into a series of tokens.
- Constants, Identifiers, Separators, Keywords and Operators are considered as tokens.
- The lexer returns all valid tokens and if it finds an invalid token such as invalid operator or a character it returns an error.
- It also removes comments and white spaces.

LEXER

- We use FLEX (Fast Lexical Analyzer Generator) to generate our lexical analyzer.
- FLEX takes in our tokens in a special file with a .l extension written in C language and generates a lex.yy.c file.
- When the generated lex.yy.c file is compiled using the C compiler we get a file named a.out which is our lexer.
- To this a.out file we give input stream and sequence of tokens is generated.

LEXER

- As mentioned to generate the lexer we use FLEX (Fast Lexical Analyzer Generator) and we write a lex file with .l extension in C language.



Using our Lexer

- The folder “Lexer” contains the following files.
  - lexer.l
  - makefile
  - input<number>.gr .
- “make” command is used to run these files and generate the output which in this case is a sequence of tokens which will be written in a separate file named output<number>.txt .
- For the detailed explanation for the commands inside the makefile see the following slides.

Lessons Learnt from this part of the Project

- After searching through several references and documentation we finally decided to use FLEX since we already had a little bit of idea in it from Compilers-1.
- Through searching we also got to learn about different methods to get lexical analyzers such as OCAML.
- Since we used FLEX we developed a better understanding of how C compiler works and how operator precedence is implemented in lexer itself.

Example

```
1
2
3   func (int) main()
4   {
5       /*
6        The edge set in the graph can be given in various ways
7        first way: n1:n2
8        An edge is in between n1 and n2
9        second way:  n1:n2:w
10       An edge is in between n1 and n2 with weight w
11      */
12      /*
13      Graph is basically an undirected graph
14      */
15
16      int a = 5$;
17
18      Graph G={
19          2:3:3,
20          1:2,
21          3:4,
22          5:1:5,
23          2:1,
24      }
25      /*
26      For directed graph, we have to use dGraph
27      */
28
29      int b = 6;
30
31      dGraph G ={
32          2:3:3 ,
33          3:1:1 ,
34          1:2 ,
35      }
36
37
38
39  }
```

Output

3 : 'FUNCTION' : 4 : 'func'

3 : 'L_PAREN' : 1 : '('

3 : 'INT' : 3 : 'int'

3 : 'R_PAREN' : 1 : ')'

3 : 'IDENTIFIER' : 4 : 'main'

3 : 'L_PAREN' : 1 : '('

3 : 'R_PAREN' : 1 : ')'

4 : 'L_BRACE' : 1 : '{'

16 : 'INT' : 3 : 'int'

16 : 'IDENTIFIER' : 1 : 'a'

16 : 'ASSIGN' : 1 : '='

16 : 'CONST' : 1 : '5'

16 : 'INVALID TOKEN' : 1 : '$'

16 : 'STM_DELIM' : 1 : ';'

18 : 'GRAPH' : 5 : 'Graph'

...

Thank You!!