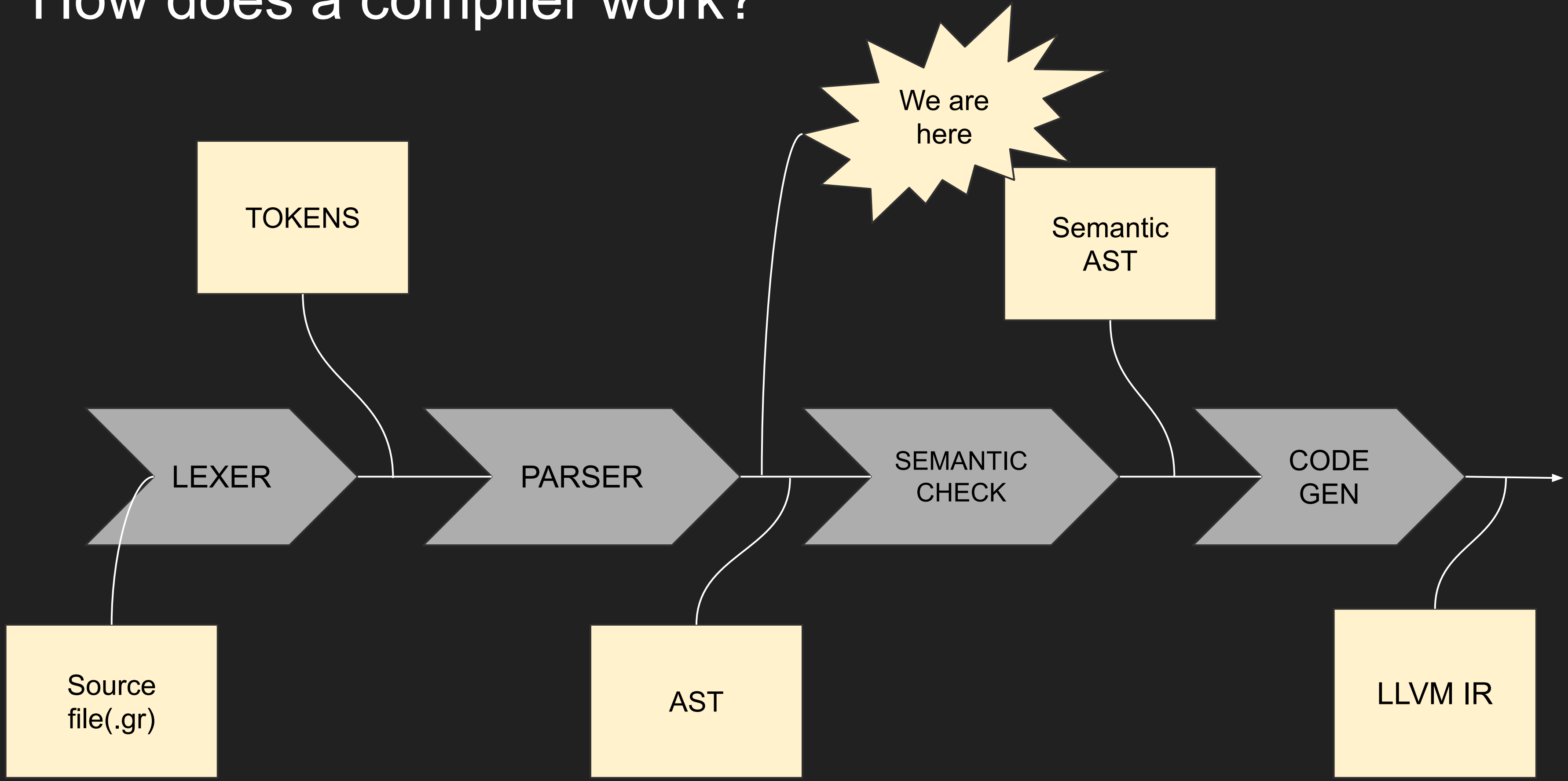
GrAlgo

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?



Parsing

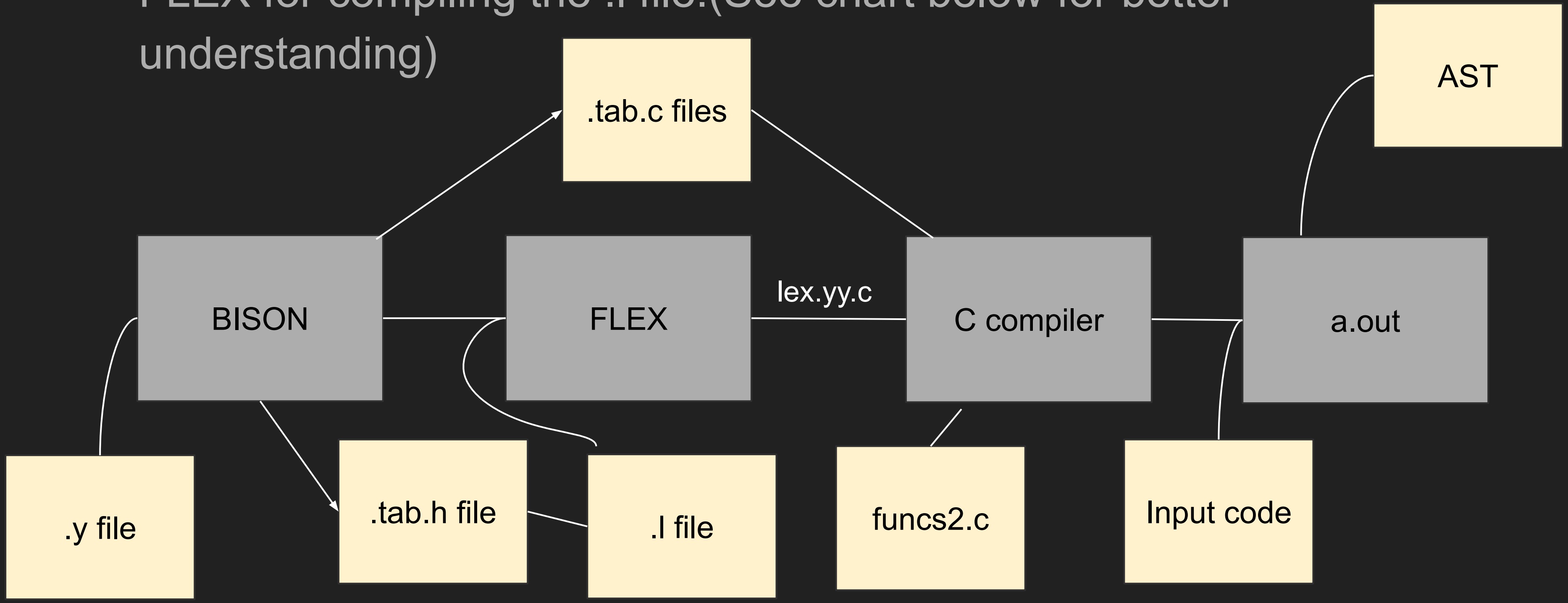
- Parsing is done to verify whether our language is following the grammar rules or not.
- It generates an Abstract Syntax Tree(AST) which is representation that conveys the structure of our source code.
- Each Node in the AST represents a construct occurring in the source code.
- Parse tree can be built using several techniques like top-down or bottom-up etc.
- We use BISON to generate a parser for our language.

PARSER

- We use BISON which is a general purpose parser generator that takes in a CFG and generates a LR Parser employing LALR(1) parser tables.
- BISON takes in our grammar in a special file with a .y which contains rules for the grammar and generates two files “parser.tab.c” and “parser.tab.h” using “-d” option
- Now we include “parser.tab.h” in the lexer file and run flex on our lexer file to get lex.yy.c file.
- We also wrote a parser-funcs.c file which contains functions that help generate AST and it’s structure.
- After getting that lex.yy.c file we compile that , parser.tab.c and parser-funcs.c using gcc and input the source code to get the respective AST.

PARSER

- As mentioned to generate our parser we use **BISON** and also use FLEX for compiling the .l file.(See chart below for better understanding)



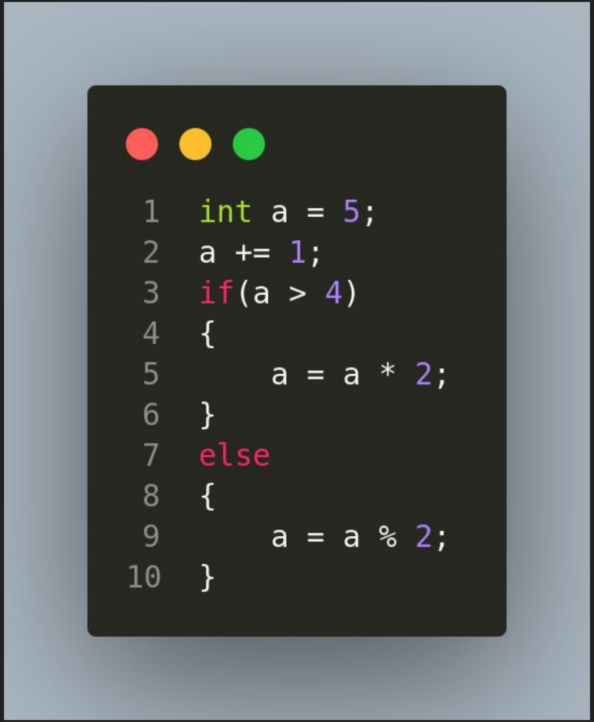
Using our Parser

- The folder “Parser” contains the following files.
    - parser.y
    - lexer.l
    - funcs.c
    - makefile
    - input<number>.gr in inputs folder (Testcases)
- “make” command is used to run these files and generate the output where in this case we obtain the Abstract Syntax Tree of the program.
- For the detailed explanation for the commands inside the makefile go through the following slides.

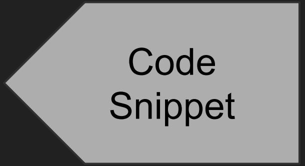
Lessons Learnt from this part of Project

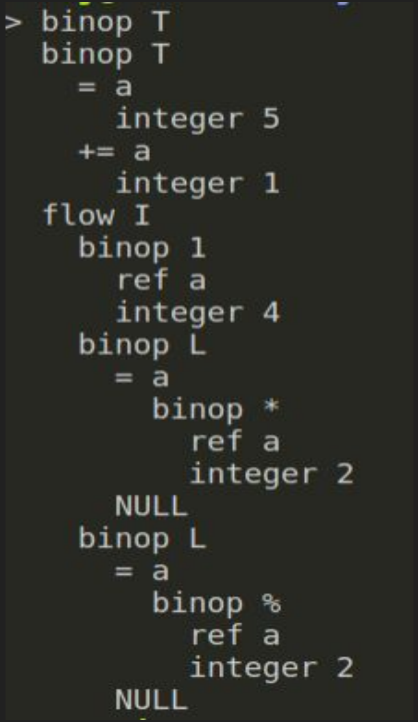
- We have learnt the Bison through Flex and Bison by O’Rielly. Bison was chosen as a parser generator as it provides a good interface for Grammar.
- We have understood how Bison works and explored different ways to generate Abstract Syntax Tree.
- Without using much predefined C Grammar we challenged ourselves to write Grammar from the start and tried various possibilities as part of our learning
- We have analysed the generated Parser with various test cases and tried avoiding all the possible shift - reduce and reduce - reduce conflicts

Example



```
int a = 5;
a += 1;
if(a > 4)
{
    a = a * 2;
}
else
{
    a = a % 2;
}
```





```
> binop T
  binop T
    = a
      integer 5
    += a
      integer 1
  flow I
    binop 1
      ref a
      integer 4
    binop L
      = a
        binop *
          ref a
          integer 2
      NULL
    binop L
      = a
        binop %
          ref a
          integer 2
      NULL
```

Thank You!!