

# SER502 PROJECT IJAR PROGRAMMING LANGUAGE

---

## TEAM 13

SALONI CHUDGAR

SANJANA MUKUNDAN

CHANDRA SHEKAR SAI ATKURI

SUDHANVA HANUMANTH RAO



# OUTLINE

---

- Introduction
- Syntax and Grammar
- Lexical analysis and Parser
- Intermediate Code Generator
- Runtime

# INTRODUCTION

- Language name: IJAR Programming Language
- Tools:

---

  - Compiler: INTELLIJ IDEA and ANTLR4
  - Runtime: Java 8
- Basic Features:
  - Supports data types- Boolean, int, string
  - Supports conditional statements and loop structures like-
    - If condition
    - For loop
    - While loop
  - Prints output on console

# BASIC FEATURES CONTD

---

- Supports basic expressions like-
  - Arithmetic ( +, -, /, \*, %, ^ )
  - Logical (&&, ||, !!)
  - Comparison ( <, <=, >, >=, ==, != )
  - Assignment ( = )
  - Ternary ( ?, : )

# SYNTAX

---

- If condition

```
if(x<5)
{
    x = x + 7;
}
else
{
    x = x - 7;
}
```

- While condition

```
while(x<5)
{
    x = x + 1;
}
```



# SYNTAX

---

- For loop

```
for (i=0; i<10; i=i+1)
{
    num = i + 3;
}
```

- For loop

```
for i in range(0,10)
{
    num = i + 3;
}
```

# SYNTAX

---

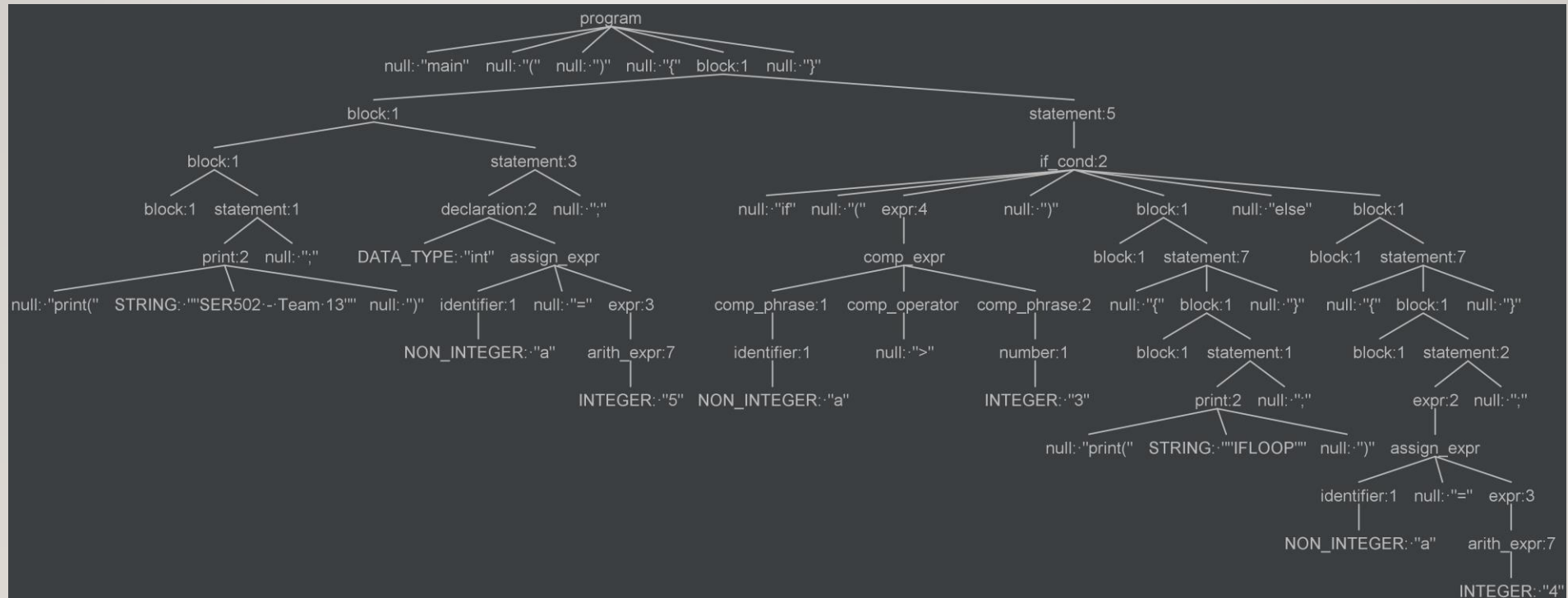
```
main()
{
    print("SER502 - Team 13");
    int a = 5;
    if(a > 3){
        print("IFLOOP");
    }
    else
    {
        a = 4;
    }
}
```

# INTERMEDIATE CODE FORMAT

```
PROGRAM  
PRINT SER502 - TEAM 13  
LOAD a  
ASSIGN 5  
IF  
COMP a 3  
PRINT  
END IF  
ELSE  
ASSIGN a 4  
END ELSE  
END PROGRAM
```



# PARSE TREE



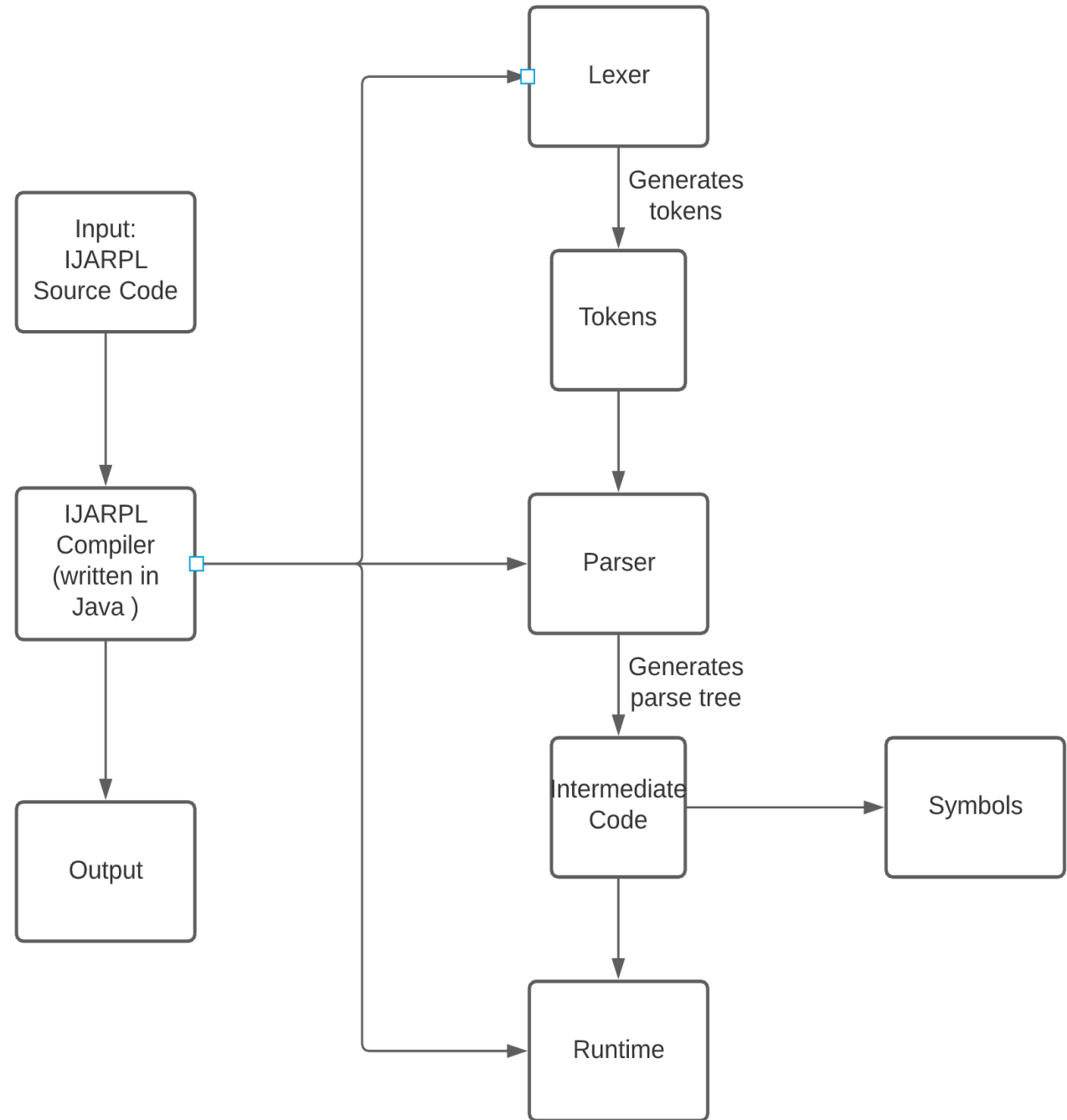
# STACK:

---

- We have used stack for :
  - Storing temporary variables
  - Combining program – program\_end, if-endif, else-end\_else, etc
  - As a data storage for the runtime file.

# ARCHITECTURE

---



# PROCEDURE

---

- First, we generated a grammar file.
- Then using ANTLR we auto-generated a lexer and parser.
- The parse tree is generated by the parser.
- By traversing the parse tree, we created the intermediate code.
- The “IJARPLListener.java” is used for traversal.
- The Antlr will generate two functions “enter ” and “exit” for every non-terminal in the grammar.
- “enter” is called on the beginning of the node while “exit” is called after everything in the node is visited.

# PROCEDURE

---

- Once, the Listener file is configured, a similar configuration is done for the intermediate code generator.
- Here, an array list is created named “codeGen” is created. Symbols from “IJARPLSymbol” are used and added to the array list based on the code used.
- This array list is then accessed and all the values printed to generate the intermediate code. The file “IJARPLIntermediateCodeGenerator” takes care of this part.



# PROCEDURE

---

- The arraylist from the intermediate code generator is then written down in a .txt file.
- The .txt file is then passed on to the execution file called “IJARPLRuntime”.
- The execution files uses two separate stacks to handle strings and numerals.
- The stacks will be iterated to convert the intermediate code to java code.
- Any errors in runtime will be displayed on the console.

# PROCEDURE

---

- The compiler is the main function which works as a controller between all the other files.
- It will get the “.IJARPL” file.
- It will send it to a lexer, get the tokens, pass the tokens to the parser, get the parse tree and then pass the tree to get intermediate code.
- Which will then be sent to executor using compiler.
- In the end the output is given by the compiler.

# THANK YOU!

---

