IMPETUS LANGUAGE

SER 502 -LANGUAGES AND PROGRAMMING PARADIGMS

SPRING 2021

TEAM 8

https://github.com/CompilerDesignTeam8/SER502-Spring2021-Team8

TEAM MEMBERS

- Shubhangi Gupta : 1219474243
- Arjun Borkhatariya : 1220063169
- Paresh Pandit : 1220832314
- Nitish Tripathi : 1219500269

Overview

We will be discussing about the following points in this presentation:

- Grammar of the language
- Information about lexical analyser
- Information about Parser and intermediate code
- Information about semantic analyser
- Sample Programs

Introduction

- Designed Language name: **Impetus**
- Tools used:
  - Compiler: Python and SWI prolog
  - Runtime: SWI prolog

- Features supported:
  - Data types: String, float
  - Looping structures : for loop, while-loop, for in range loop
  - Conditional constructs: if-else-statement
  - Arithmetic Expressions : +, -, * , /
  - The language supports syntactic sugars like +=, -+, *=, /= ,++, --
  - Comparison operators ==, <=, >= ,<, >, !=, not(!) are supported
  - Output command to show results on console : print << “value”
  - Newline supported using : endl
  - Comments can be declared by #
  - Extension: .ipt for source code and .iptok for intermediate code

Keywords in the language

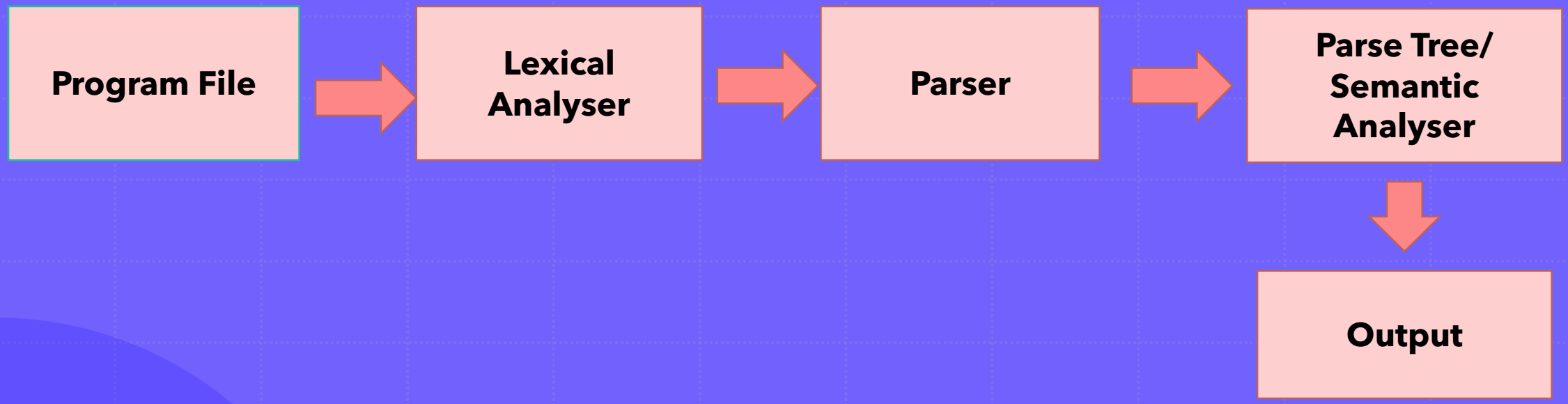
- float
- string
- if
- else
- main
- print
- for
- while
- true
- false

GRAMMAR

The Impetus Grammar supports the following:

- Block
- Statements
- Declarations
- Commands
- Boolean
- Expressions
- Datatype
- Print statement

Code Flow


Program File
Lexical Analyser
Parser
Parse Tree/ Semantic Analyser
Output

LEXICAL ANALYZER

- Input - Program file
- Input file extension - .ipt
- Output – Tokens
- Output file extension - .iptok
- The lexical analyzer breaks down the program into tokens which is sent as input to the parser.

Lexical analysis example

mycode.ipt

```
main{
    int i = 0;
    int j = 2;
    for(int i = 0; i < 20; i++){
        print << 'This is string ' << a << endl;
    }
    if(a>b){
        print << 'what can i say I'm just smart';
    }
    while(b>a){
        a += 1;
    }
}
```

mycode.iptok

```
[main, '{', int, i, =, 0, ;, int, j, =, 2, ;, for,
'(', int, i, =, 0, ;, i, <, 20, ;, i, +, +, ')', '{',
print, <, <, ', This is string, ', <, <, a, <, <, endl,
;, '}', if, '(', a, >, b, ')', '{', print, <, <, ',
what can, i, say, I, ', m, just smart, ', ;, '}',
while, '(', b, >, a, ')', '{', a, +, =, 1, ;, '}', '}']
```

PARSER

As part of the syntactical analysis, we have developed a parser for the language.

Parser is written using SWI prolog.

The program for parser includes a series of rules and facts as part of the knowledge database.

The input for the parser will be the tokens generated from the lexical analysis.

The tokens are in the form of a list of strings.

This list is processed through a number of rules and facts written in prolog and checked for validity.

If a token does not satisfy any of the rules or does not belong to the facts, then the program outputs a failure scenario which in turn fails the compilation process.

If the input sequence of tokens is valid which will be denoted by the parser outputting “true” then the input sequence is ready for the next leg of the compilation process that is the parse tree generation.

Parse tree generator is the site where intermediate code is generated for the program.

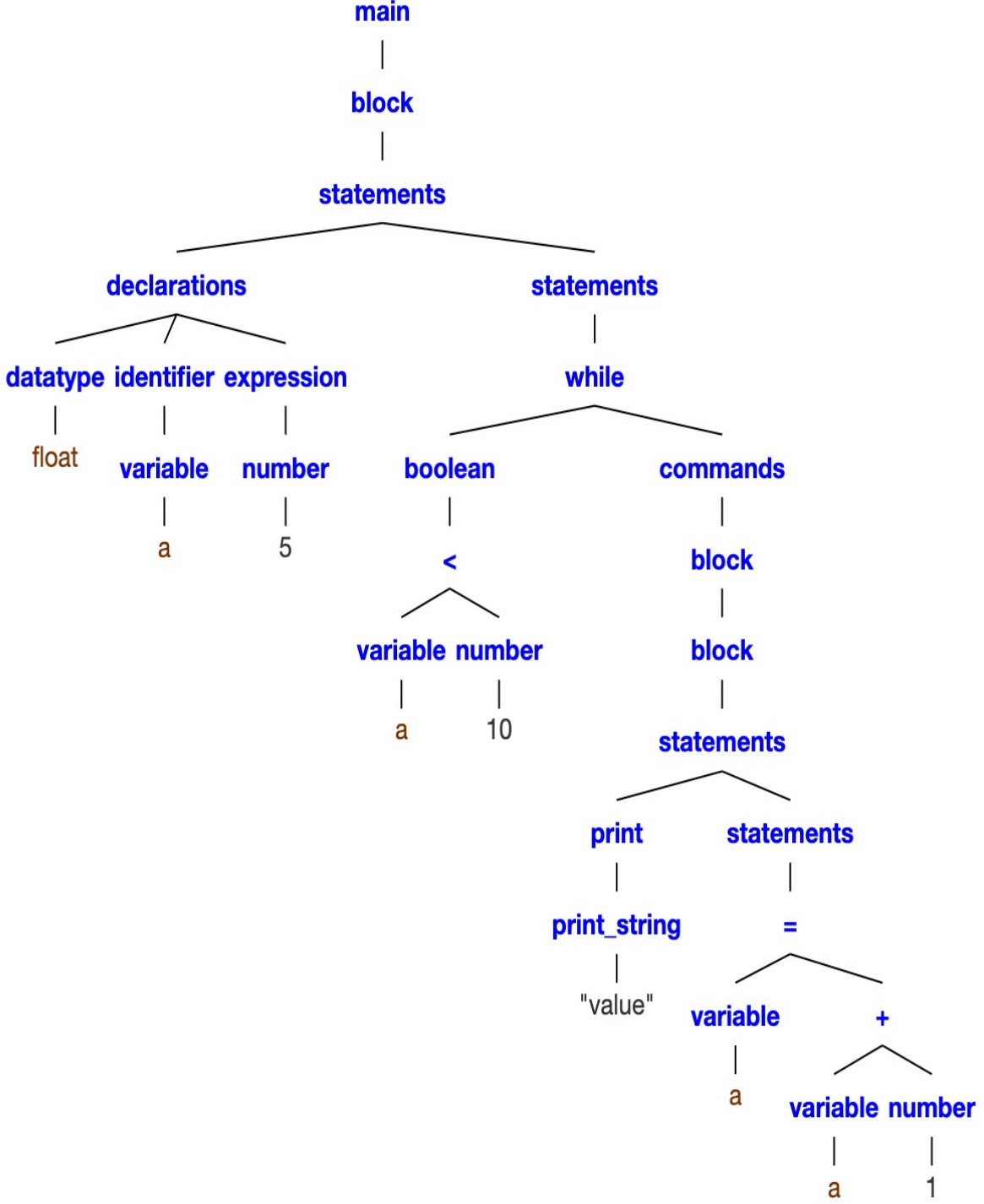
Parse tree generator

- This part of the semantic analysis is responsible for generating parse trees/intermediate code of the existing code.
- We have used SWI prolog for the implementation of parse tree generator.
- Once the code has gone through the parser successfully the parse tree generator will take the sequence of tokens.
- The output of this program will generate a tree where each token is represented under the structure that they belong to.
- This tree will be further used in the semantic analyser or evaluator to generate the result of the **impetus** code**.**

Parse Tree Sample run

Code:

```
main
{
  float a = 5 ;
  while(a<10)
  {
  print<< "value";
  a = a + 1 ;
  }
}
```



SEMANTIC ANALYSER

- The final phase of the compilation process involves evaluation of the parse tree.
- This is the stage where the purpose of the impetus code is being delivered in form of an output.
- Technology used for evaluator is SWI prolog.
- The key aspect of evaluator is manipulating an environment.
- Based on the conditions in the code the environment is being updated (added to or deleted from). There are methods like update and lookup which are being used to look for and update the set of key value pairs in the environment.
- The Final environment is being used to calculate the result of the program.

SAMPLE RUNS

mycode3.ipt | impetus_tokenizer.py

```
1  main{
2      float a = 5;
3      for i in range(a){
4          print << 'this is i' << i << endl ;
5      }
6  }
7
```

```
Please enter .ipt file name from data dictionary :- mycode3.ipt
this is i 0
this is i 1
this is i 2
this is i 3
this is i 4
```

SAMPLE RUNS

```
main{
    float a = 1;
    float b = 2;

    while (a < 10) {
    print << 'a ' << a << endl;
        b = 2;
        while (b < 10) {
            print << 'b ' << b << endl;
            b++;
        }
        #a++;
    }
}
```

```
a 1
b 2
b 3
b 4
b 5
b 6
b 7
b 8
b 9
a 1
b 2
b 3
b 4
b 5
b 6
b 7
b 8
b 9
```

SAMPLE RUNS

```
main{
    float a = 5;
    float b = 2;

    # demo of for loop
    for(float i = 0; i < 3; i++){
        print << 'val of i ' << i << endl;
    }

    # demo of if condition
    if(a>b){
        print << 'still a is greater than b and value of a is' << a << endl;
    }

    # demo of while loop
    while(b<a){
        print << 'still b is greater than a and value of b is' << b << endl;
        b += 1;
    }
}
```

```
Please enter .ipt file name from data dictionary :- mycode3.ipt
val of i 0
val of i 1
val of i 2
still a is greater than b and value of a is 5
still b is greater than a and value of b is 2
still b is greater than a and value of b is 3
still b is greater than a and value of b is 4
```

FUTURE SCOPE

- Arrays
- Local scope
- Global scope
- More datatype support
- User input
- Functions

THANK YOU.