CS 335 Semester 2022–2023-II: Project Milestone 2

Group members:
Priyanka Jalan (190649)
Piyush Agarwal (190600)
Gargi Naladkar (200371)

March 26, 2023

Using Lex and Bison, we developed a scanner and parser that accepts any valid Java program with the features listed below as input and outputs a DOT script representing the AST(abstract syntax tree) of the input program. The DOT script, when processed by the Graphviz tool called dot, will produce a postscript file with the diagram of the parse tree. We incorporated the scanner and parser specifications that are stated in the issue statement for milestone 1.

In milestone 2, we implemented support for symbol table data structure to the parser we have developed in milestone 1. We also performed semantic analysis and did type checking including type casting. And finally we converted our input into 3-address code (3AC) code (intermediate representation).

The specifications of our implementation are listed below:

Compilation and Execution:

1. Download the cs335 project zip file. Extract the contents.
2. Move to the src directory by using the following command in your teminal.

```
    $ cd milestone1/src
```

3. To generate executable and compile the lexer and parser files, use the following command:

```
    $ make
```

   The above command generates the following files:

   (a) myASTGenerator : an executable file.
   (b) parser.output : descriptions of various states of LR.
   (c) parser.tab.c : parsing program in C
   (d) lex.yy.c : lexer program in C
   (e) parser.tab.h : header file

   Makefile contains the following commands:

   (a) For parsing using "make" command:

```
    bison -dtv parser.y
    flex lexer.l
    g++ -w lex.yy.c parser.tab.c -o out
```

(b) For cleaning using ”make clean” command:

```
    rm myASTGenerator
    rm parser.tab.c parser.tab.h parser.output
    rm lex.yy.c
```

(c) For testing using ”make test” command:

```
  ./myASTGenerator ../tests/test_1.java
  ./myASTGenerator ../tests/test_2.java
  ./myASTGenerator ../tests/test_3.java
  ./myASTGenerator ../tests/test_4.java
  ./myASTGenerator ../tests/test_5.java
  ./myASTGenerator ../tests/test_6.java
  ./myASTGenerator ../tests/test_7.java
  ./myASTGenerator ../tests/test_8.java
  ./myASTGenerator ../tests/test_9.java
  ./myASTGenerator ../tests/test_10.java
```

4. A typical session for running the test file with address ”../tests/test_1.java” which creates a directory with name ”output” containing with 3ac for all the functions with the file name ¡Class Name¿.¡Function Name¿.csv and symbol table too, use the following command:

```
    ./myASTGenerator ../tests/test_1.java
```

5. The following options are supported for running the executable file:

```
usage: ./myASTGenerator [options] file
Options:
--help                  To print this message.
--input=<file name>     You can use it to specify the input file name.
                        By default, the name of the input file can be
                                                                 specified
                                                                 directly.
--output=<file name>    You can use it to specify the output file name.
                        By default, the name of the output file is 'output.
                                                                 dot'
Do not use --input=<file name> flag if you specify the input file name
                                          directly.
--verbose               It outputs the stack trace
```

6. To Render the AST generated, use the following command:

```
    dot -Tps out.dot -o image.ps
```

Features:

We have implemented following features and created 10 java programs for testing -

- Support for primitive data type
- Variables and Expressions
- For and While loops

- Multidimensional arrays
- Support for print() and println()
- Support for Strings(addition of strings with strings, int, boolean, char, float)
- Supported text block, integral data types(byte, short, int, long), floating data types(float, double), this keyword

- Variable Declaration supported via format:
  int a, b = 10, c;

- Support for operators $(+, -, *, <, >, \leq, \geq, <<, >>, >>>, ||, \&\&, |, \&, ==, !=, ?:)$
  1. Complement Operators: $(\sim, !)$
  2. Additive operators: (+ , -)
  3. String concatenation operator (+)
  4. Shift Operators: $(<<, >>, >>>)$
  5. Conditional Operators:$(\&\&, ||)$
  6. Relational Operators: $(<, >, <=, >=)$ Numeric comparision.
  7. Equality Operators: (==, !=)
  8. Ternary Operator: (? :)
  9. Multiplicative Operators: (*, /, % (integer and floating))
  10. Assignment Operators: $(=, *=, /=, \%=, +=, -=, <<=, >>=, >>>=, \&=, \wedge=, |=)$
  11. Bitwise and logical Operators: $(\&, |, \wedge)$

- Primitive data type Array support with multi-dimensional array(applicable for greater than 3 too) declaration using new keyword :
  int arr[] = new int[10];

- Support for Unary Operation:
  ++ and - - on numeric types(both post and pre)
  +, - as unary operator on numeric types.
  ! works on boolean type only.
  ∼ works on numeric integral types.

- Class object declaration using new keyword. e.g.
  class tree { }
  public static void main(){
  tree obj = new tree();
  }

- Support for Method Declaration(Method Declaration after the main function and called inside the main method.)

- Support for object method invocation, field access.

Generation of 3-Address code:

- On execution the compiler generates files containing the 3AC code for each declarations within the class body.
- Each file corresponding to that declaration contains the 3AC code for every statement in that declaration. Each statement is represented using addresses and operators.
- For conditional statements jumps are defined using ”goto” and ”IfFalse” and ”IfTrue” are used correspoding to false and true values of the statements. .For ”for” and ”while” loops also jumps to different states defined.
- For code x + y, if x is an int and y is a float, ”cast_to_float” is used to represent type casting.
- ”alloc_memory” used to represent allocation of memory of any array.
- Allocating memory for the array creation.
- **recursion, return, break, continue** statements working in 3ac
- Conditional statements working fine in 3ac
- Stack operations push, pop, call supported.