Milestone 1

README

CS335 Compiler Design

1 Tools Used

- We have used the **GraphViz** tool to visualise AST for our grammar. This required knowledge of DOT language and the dot tool.
- We used the grammar for JAVA mentioned on the website here.

2 Command Line Options

- – –**input** Add this flag for specifying a input file to the parser. This is a **required** flag. Example:

```
./main --input=input.java
```

- – –**output** Add this flag for specifying a output file to the parser which would contain the output i.e a AST in graphical form. This flag is optional. Default value is **"output.dot"**. Example:

```
./main --input=input.java --output=result.dot
```

- – –**help** Add this flag for reading the rules regarding running the commands. This flag is optional. Example:

```
./main --help
```

- – –**verbose** Add this flag for switching on the debug mode in the parser. This flag is optional. Example:

```
./main --input=input.java --output=result.dot --verbose
```

3 Compilation Instructions

After extracting the zip folder of the submission. Open the terminal and execute the following to compile the parser.

```
    cd milestone1/src
```

Now after being in the same directory as the source files of the compiler we run the make command as follows.

```
    make clean
    make compile
```

4 Execution Instructions

To execute run the following command in the with the same rules specified in Section #2.

```
    ./main --input=BubbleSort.java
    dot -Tpng <output_file_name> -o AST.png
```

Here the " <output_file_name >" is name of the same file as specified with the –output file , if no output file was specified then the default output file name is **output.dot** .

To get the x86 code from the 3AC code, run the command:

```
make x86
```

To run a specific test case, you can execute:

```
./run.sh <test-case-number>
```

For eg.

```
./run.sh 1
```

To test the X86 code, execute command:

```
make run
```

5 Output Files

The output files i.e the CSV files of Symbol tables and 3AC instruction is stored as **.csv** files and **.txt** files respectively in the folder **milestone2/src/output** along with the x8 code saved as **.s** file. The folder is automatically created on compilation and files are saved after execution.

The Symbol tables are saved as : **Function- <function_name >.csv** insidethier respective class folder
The 3AC is saved as : **3AC.txt**
The x86 code is saved as : **x86.s**

Each csv files stores the name of function, the number of parameters in that function and the return type of that function. It further stores the variables directly declared inside the scope of that function. If **n** is the number of parameters of that function, then the first **'n'** entries are the parameters of the function. The variables have their types, lexeme, token and line number stored in the table.

6 Bonus Optional Features added

- Interfaces
- Type Casting : We added the support for type casting and we save the changed type of a variable in the symbol table whenever its type is changed and its entry is found.
- Imports and Packages