CS 335 Semester 2023–2024-II: Milestone 1

A. Atulya Sundaram, 210001
Anuj Singhal, 210166
Goutam Das, 210394

Compilation and Execution Instructions

The `milestone1` directory has 3 subdirectories

- `src`: containing the project's source files, i.e. the scanner(`lex.l`), parser(`parser.y`) and a `makefile`.
  To compile the code, you can simply use the given `makefile`, make sure that `flex`, `bison` and `g++` are installed on the system

  ```
  $ make all
  ```

- `tests`: Include 5 test programs for testing the parser
  - `test1.py`: Takes 2 array, sorts them using insertion sort and merges them and removes duplicates
  - `test2.py`: Takes an array of strings, sorts them in reverse lexographical order and count the number of vowels
  - `test3.py`: Makes classes for squares, rectangles and circles and finds their area and perimeter
  - `test4.py`: Given an array, find triplets with maximum sum, bitwise or, bitwise and etc.
  - `test5.py`: Implemented class for a space travel system with different methods
- `doc`: Includes documentation of the project (i.e. this file)

The implementation has been tested to successfully compile and execute on

- `flex 2.6.4`
- `bison 3.8.2`
- `g++ 11.4`

`make` command creates an executable named `parser`, to parse a file using parser, the usage instructions are as follows:

```
USAGE:
[-input <path-to-inputfile>]: specify the file path to parse, if not given, takes
input from stdin
[-output <path-to-outputfile>]: specify the file path to write the ast, if not
given, defaults to ast.dot
[-verbose]: prints verbose messages to stdout, error messages are anyways printed
[-help]: print usage instructions
```

An example command to run parser on file named `test1.py` and generate `graph.dot` file in a verbose manner is:

```
$ parser -input test1.py -output graph.dot -verbose
```

To compile the generated DOT file into a pdf, use the `graphviz` tool as follows

```
$ dot -Tpdf -Gordering=out ast.dot -o graph.pdf;
```

The generated pdf contains the abstract syntax tree of the given input code.

Output Format

The output is a dot file which describes the abstract syntax tree of the input code, here are a few details about the tree generated:

- The root node is labelled "input", all the statements are children of root, with "ENDMARKER" as the rightmost child (kept to denote end of file)
- As generally written, the root node describes the operation and children describes the operands
- The children node of "while loop" are the condition and the body of the loop, nodes for other constructs like "function definition" and "if statement" are designed in a similar way
- The statements in the body of a function/while/for loop are made children of a node labelled as "statements"
- An expression enclosed in parenthesis is made child of a node labelled "()", and same for "[]"

Error Handling

In case of errors, the error message is printed to stderr, the abstract syntax tree is generated from the token seen till now but it may contain extra, disconnected nodes for due to incomplete productions.
The errors reported by the parser are as follows.

- In case of an invalid token which is not in the language, the scanner can report the error as follows:

  ```
  Error: Unrecognized token <token> at line <line_number>
  ```

- In case of a syntax error, the parser reports the error along with the token at which the error was detected as follows

```
SYNTAX ERROR: Unexpected token: <token> at line <line_number>
```

Also based on the type of token, it can print the corresponding token type as well like

```
SYNTAX ERROR: Unexpected string literal: <string> at line <line_number>
OR
SYNTAX ERROR: Unexpected indent at line <line_number>
```