CS335: COMPILER DESIGN

Milestone 1

HDS PYTHON COMPILER

Harsh Bihany (210406)
Danish Mehmood (210297)
Siddhant Suresh Jakhotiya (211030)

March 3, 2024

1 Running the files

The implementation of Parser and Lexer is in Flex+Bison, and is written in C++. Further, DOT is used to create a graphical representation of the AST and exporting it to a PDF file. User should install all the above (if not already installed) by running the following commands:

```
$ sudo apt-get update
$ sudo apt-get install flex
$ sudo apt-get install bison
$ sudo apt-get install graphviz
```

A script file `pyrun.sh` has been made to help run the program against a test case. The command line options provided by our program are as follows:

- `-i` or `-input`: Filename from which the compiler reads the Python program.
- `-o` or `-output`: Filename to which the compiler outputs the `DOT` file.
- `-v` or `-verbose`: Prints additional checkpoints to show progress.
- `-h` or `-help`: Shows the usage for the script.
- `-c` or `-clean`: Performs `make`, with given options (if any) and then performs `makeclean`.

Each of the commands mentioned above is optional and can be utilized in any sequence. The script takes `../tests/test1.py` as default input and `graph.pdf` as the default output, does not print additional messages and does not perform `make clean` after execution.

An example of a successful default execution is as follows:

```
$ ./pyrun.sh
```

An example of successful execution using the command line options provided above against the test case `test3.py`, the output being redirected to `graph1.pdf` and with additional debug messages, is as follows:

```
$ ./pyrun.sh -input ../tests/test3.py -o graph1.pdf -verbose -c
```

2 Codebase

The codebase can be found at our GitLab ID `hds-cs335`, inside the repository `python-compiler-2024`. The whole codebase is inside the directory `milestone1` which is on the `main` branch of the repository mentioned before. The file structure of the directory `milestone1` is as follows:

```
milestone1/
|-- src/
|   |-- include
|   |       `-- node.hpp
|   |-- Makefile
|   |-- main.cpp
|   |-- node.cpp
|   |-- pylex.l
|   |-- pyparse.y
|   `-- pyrun.sh
|-- tests/
|   |-- test1.py
|   |-- test2.py
|   |-- test3.py
|   |-- test4.py
|   `-- test5.py
|-- doc/
|   `-- hds_milestone1_report.pdf
`-- personal_docs/
```

The same file structure (except the file `personal_docs`) has also been following in the zip file uploaded to **Canvas**.

Description for different files is as follows:

- **pylex.l** : This `Flex` file houses the code for the lexical scanner. It scans the input file and passes the `tokens` to the `parser`. It also keeps track of the line number for error reporting.
- **pyparse.y** : This `Bison` file contains the code for the parser. It drives the lexer and generates the `Abstract Syntax Tree`.
- **pyrun.sh** : This file is the `script` that helps run the compiler against a testcase. It also implements the command line options `-input`, `-output`, `-verbose` and `-help`. It concludes with a `make` command that executes the `Makefile`, which then runs the program with the supplied arguments to generate the output PDF.
- **main.cpp** : This `C++` file has the code to drive the `parser` and generate the `dot` script.
- **node.cpp / node.hpp** : These files house the function definitions of `struct node` methods and functions for `AST` generation.
- Additionally, the files `test1.py`, `test2.py`, `test3.py`, `test4.py`, `test5.py` are the test cases provided by us as asked in the problem PDF.

3 Augumentations

The Python documentation offered thorough lexical and grammar specifications. The grammar, being LALR, had no conflicts. We tailored these grammar and lexical specifications to meet the requirements specified in the problem PDF by including only those productions which were necessary to implement said functionality.

4 Error Handling

All the expected language features as asked in the problem statement PDF have been supported. Following errors have been handled as well:

- Indentation errors: Only even number of spaces are considered for indentation. Any odd number of indentation spaces will lead to error. `Tabs` will also lead to error.
- Unterminated string
- Syntax error in the assignment operator and an `if` statement missing a test condition.

The program exits after the first error encountered and reports the line number along with appropriate error message. In the above mentioned scenarios, our error messages mirror those produced by the actual Python compiler.

5 AST from Parse tree

The `parser` is responsible for generating the parse tree. Some nodes are pruned from the parse tree generated by the `parser` according the following rules (in same order as listed):

- Removed all non-terminals that were not part of the original Python grammar but were added by us for the purpose of writing the Bison code.
- Expressions in the AST have been manipulated to facilitate easier 3AC (Address Code) generation by grouping binary operands with an operator.
- In Python3, unlike in C, chained comparison operators are assessed sequentially. For instance, the expression `a == b > c` is interpreted as `a == b` **and** `b > c`. This behavior has been integrated into our AST.
- Any non-terminal evaluating to ϵ is also pruned.
- All **delimiters** have been pruned except $\rightarrow$.
- All non-terminal nodes with a single child have been pruned for a simpler tree.

The Abstract-Syntax-tree generated can be found in the file `graph.pdf` (by default), generated after running the program against a testcase.

6 References

- Python docs for Lexical specifications
- Python docs for grammar specifications
- Graphviz docs