Pythoneers: Milestone 2 Report

Chitwan Goel	Shrey Bansal	Talin Gupta
Roll No. 210295	Roll No. 210997	Roll No. 211095

Code Base Structure

```
milestone2
├── doc
│   ├── Effort Sheet.txt
│   ├── milestone1.pdf
│   ├── Milestone 1 Report.pdf
│   ├── milestone2.pdf
│   └── Milestone 2 Report.pdf
├── include
│   ├── global.hpp
│   └── symbtab.hpp
├── scripts
│   ├── compile.sh
│   └── run.sh
├── src
│   ├── helper.cpp
│   ├── lexer.l
│   ├── new_grammar.txt
│   ├── parser.y
│   └── symbtab.cpp
└── tests
    ├── test1.py
    ├── test2.py
    ├── test3.py
    ├── test4.py
    ├── test5.py
    └── test6.py

6 directories, 20 files
```

Requirements

To run the program, you will need the latest versions of `bison` and `flex`.

Usage

We have provided a script `run.sh` to run the program in the `scripts/` directory. The output `csv` files (for symbol tables) and `txt` file (for 3AC) will be generated in the `src/` directory. If no output file name is mentioned, the default output file is `3AC.txt`. Run the following commands:

```
cd scripts
./run.sh --input=<input_file_path> [options]

The following options can be used
--output=<output_file_path>     : Output 3AC file specification (default
                                  is 3AC.txt)
--help                          : Instruction regarding usage
                                  instructions and options
--verbose1                      : Prints our custom debug statements
                                  while reducing a production
--verbose2                      : Prints the complete stack trace of the
                                  parser execution
```

Providing the input file is necessary via the `--input` flag. The input file must have `.py` extension. The input file path should be relative to the `src/` directory. The program will terminate and throw an error if the input file is not specified.
Alternatively, you can use the script `compile.sh` to generate an executable `parser` and run it as follows:

```
cd scripts
./compile.sh
./parser --input=<input_file_path> [options]
```

Optional Features Included

We have supported the following extra features:

- Chaining of object attributes (eg. `obj.a.b`, `obj.a.b()` and `obj.a().b`)
- Implicit and Explicit Line Joining

Printing Symbol Tables

For each scope, the symbol table has its own CSV. The CSVs appear in `src/` directory.

- Global symbol table (`st.csv`): Has sections for classes, functions, and variables. The token specifies whether it's a class, function, or variable. The return types of functions are also mentioned here. Parent classes are also mentioned in the enclosed parenthesis.
- In the global symbol table, by default, there are 4 functions for print, each taking different arguments. There is also the len function. Range has not been implemented as a function; we have treated it as a keyword.

- Class symbol table (`st_<class_name>.csv`): Has entries for functions and variables within the class (names, types (or return type in case of functions))
- In the class symbol tables, we don't store the attributes and methods of the parent symbol table. Instead, we store only the pointer to the parent class, and at the time of usage of an attribute or invocation of a method of the parent class, we do a lookup in the class inheritance hierarchy.
- We have assumed that the attributes of a child class are (implicitly) laid after the attributes of the parent class. Hence, the offsets of the attributes in the child class start from the size of the parent class.
- Function symbol table (`st_<class_name>_<func_name>.csv`): Has entries for variables and formal parameters (name, type, and line number).

Three Address Code Generation

- The instruction number indexes each instruction/statement in the 3AC.
- Temporaries start with `__t`.
- Name mangling has been used, @ is used as the separator.
- `call strcmp` in the 3AC refers to the GNU implementation of strcmp, and `call print` refers to the syscall to print to stdout.
- For `jump` / `goto`, we use the instruction number as labels.