CS335A: COMPILER DESIGN
Milestone-2

Rishabh Arijeet-210842
Shishir Gujarey-210977
Prachi Choudhary - 210732

March 2024

1 Tools Used

We have used flex for lexer, bison for parser, both written in C++ and graph visualization tool called Graphviz for AST visualisation.
Python 3.8 is used to minimize the conflicts and grammar complexities.

2 Running the files

There is a Makefile inside the src directory and the executable **ast** can simply be made by running **make** inside the src directory.

The command line options are:

- **-h** or –**help**: Prints the manual for using the **ast** executable.
- **-i** or –**input**: Takes the input file.
- **-o** or –**output**: Takes the output dot file.
- **-v** or –**verbose**: Prints the derivation(parse tree) of the input.

The default input file is **test1.py** inside the tests directory and default output file is **graph.dot**. Now, the following command generates a PDF containing the visualization of the AST.

$ dot - Tpdf graph.dot -o graph.pdf

A successful execution is given below:

$ make
$ ./ast -i testcase1.py -o tree.dot -v
$ dot - Tpdf tree.dot -o tree.pdf

The symbol tables are generated in .csv format. The global symbol table is in the file **symtab.csv**. A new symbol table is created everytime a new scope is encountered. The files for these symbol tables are named according to their scope. E.g. the symbol table for a function named 'func' which is defined in the class 'abc' will be in the file **symtab_abc_func.csv**.

The dump of the 3AC code is in the file **tac.txt**.

3 Creating Symbol Tables

The symbol tables contain the scope, name, data type and line numbers of the symbols. The scope of every new symbol table is determined according to its parent scope and its position in the parent scope. If a parent scope has scope s and has two child scopes, the first one will have scope s.1 while the other will have s.2. The scope of the global symbol table is kept blank.

4 Semantic Analysis

The following semantic analysis is taken care of while parsing:

- Identifiers are used within the scope of their declarations.
- Type checking all computations involving variables.
- Matching type and number of arguments of a function.

5 3AC

For generating the 3AC representation, we have used the quadruple data structure. A quadruple is generated for each node, which is then passed on to its parent. For each 3AC, we have used its line number as the label. The 3AC code is in the following forms:

- The new variables generated are of the form _t3, where 3 is the rank of the variable generated
- For function declarations, begin_func and end_func are used at the beginning and end of the functions respectively.
- For class declarations, begin_class and end_class are used at the beginning and end of the class respectively.
- For function invocations, call_func is used.
- For creation of an object, call_ctor is used
- For lists, appropriate memory is allocated using alloc_mem and then list elements are allotted corresponding memory location.
- For return statement, the parameter to be returned is first pushed using push_param and then return is used.
- pop_param and push_param are used for the parameters used accordingly.
- During function or constructor call, necessary stack operations for allocating, storing and restoring data are performed using stack_pointer.