CS335A: COMPILER DESIGN
Milestone-3

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

April 2024

1 Tools Used

- **Flex**: We have used flex for lexer.
- **Bison**: We have used bison for parser.
- **C++**: All codes aare written in C++.
- **Graphviz**: Graph visualization tool called Graphviz is used for AST visualisation.
- **Grammar**: Python 3.8 is used to minimize the conflicts and grammar complexities.

2 Features supported

- Primitive data types (e.g., int, float, str, and bool)
- 1D list (ignored dictionaries, tuples, and sets)
- Basic operators:
  - Arithmetic operators: +, -, *, /, //, %, * *
  - Relational operators: ==, !=, >, <, ≥, ≤
  - Logical operators: and, or, not
  - Bitwise operators: &, |, ˆ, ˜, ≪, ≫
  - Assignment operators: =, +=, -=, *=, /=, //=, %=, **=, &=, |=, ˆ=, ≪=, ≫=
- Control flow via if-elif-else, for, while (ignored pass, do-while and switch)
  - Support for iterating over ranges specified using the range() function.
- Support for recursion
- Support for the library function print() for only printing the primitive Python types, one at a time
- Support for classes and objects, including multilevel inheritance and constructors. Ignored multiple inheritance (i.e., a class can have only one parent class).
- Methods and method calls, including both static and non-static methods
- Static polymorphism via method overloading

3 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 asm 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 containing the x86_64 assembly code is **asm.s**. The default .dot file containing the AST is **graph.dot**.

The following command generates a PDF containing the visualization of the AST.

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

A successful execution for generating and executing the generated x86_64 assembly code is given below:

$ make
$ ./ast -i testcase1.py -o asm.s
$ gcc -c asm.s -o test1.o ; gcc test1.o -o test1 ;
$ ./test1

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**.

Corresponding to every 3AC instruction, asm code is generated and written to **asm.s** file.

4 Unsupported Features

- Member functions and class constructor support only assignments and return statements.
- x86_64 code for lists is not supported. Lists are supported only until generation of 3AC code.
- Break and Continue statements are not supported.
- Power operator and unary operators and are not supported.
- Print statements do not support function calls.

5 Individual Contributions

Member	**Username**	**Roll Number**	**Total Contribution**
Shishir Gujarey	shishirg21	210977	36.22
Rishabh Arijeet	rishabh21	210842	36.22
Prachi Choudhary	prachic21	210732	27.55