CS335 Project - Milestone 3
Group 32

Shubham Patel	Harsh Murdeshwar	Shubham Anand
210709	210641	211020

April 18, 2024

1 Features

1.1 Required

The project provides support for the required features that had to be supported by the complier we have developed for the purpose of the milestone 3.

1.2 Additional

We have additionally provided support for the following features:

- Generalised `range` Function:
  There was a constrain set on the argument for the `range` function that only constants will be used. We have additionally provided support for any type of integer expression as an argument to the `range` function.
- Method-Calls via Class Name:
  In general python, the methods of a class can not only be invoked by objects of the class, but also they can be invoked by the name of the Class. Our implementation additionally supports this feature of python.
- Chained Attribute Accessing:
  The project had a constrain that the `.` operator for accessing attributes of an object will not be used more than once. We have supported a generalised functionality which puts no such constrains of the number of times the `.` operator can be used sequentially.
- Delayed Assignment of Lists:
  The milestone had a constrain that whenever a list is declared, it will be assigned a value. We have relaxed this constrain and permitted the list variable to be declared first and assigned later before usage.
- `rstings`:
  While in milestone1 it was mentioned that special types of strings are

not required to be supported. We have extended the allowed strings to included the `rstings` which makes it more expressive.

1.3 Restrictions

The following are the minor restrictions we impose:

- Function names can't be of the form "STRn" or "Ln" since we have used these to mark labels in the assembly.
- In the excecutable which has been generated, we have disabled address space randomisation for simplicity.

2 Usage

We have used a wrapper script so that the process becomes easier. No manual changes required. The inputs to the script will be the Python source-code file and the prefix for the output files. The script will first `make`, and then produce the output files if the program is correct. The output files produced will be a `.txt` consisting of the 3AC and a `.csv` consisting of all the symbol tables. An `.s` file will be generated. The script will then finally compiler and the run code. The script name is `make_py.sh`.
We have supported the options:

- `-v`, `--verbose`: To output the details of nodes being created as and when they are made.
- `-i`, `--input`: To specify the input python program file.
- `-o`, `--output`: To specify the output prefix for the files.
- `-h`, `--help`: To get basic information on the usage of the `make_py.sh`.

Syntax:
`./make_py.sh [options] <inputfile> <outputprefix>`

Example Usage:
```
./make_py.sh test1.py test1_output
./make_py.sh -v --input test1.py -o myoutput
./make_py.sh --verbose --input test1.py output1
```

3 Modifications to 3AC

- Milestone 3 testcases would not require us to deal with floats so we have ignored floats. Also as it has been told that global variables do not have to be supported, so we have followed the same.

- For milestone2 we took size of int as 4 bytes and bool as 1 byte. We changed this to 8 bytes, 8 bytes to make register operations simpler.
- For milestone 3 we have assumed that main() will be directly as the first function. We have assumed that the if __name__ == "__main__": block will only call main since global variables are not allowed.

4 Tools Used

The following tools (along with their version numbers) were used for this milestone:

- Flex 2.6.4
- Bison 3.8.2
- g++ (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
- GNU Make 4.3

If these tools are not already installed, we can install them using:

```
sudo apt update
sudo apt install flex
sudo apt install bison
sudo apt install make
sudo apt install g++
```

5 Effort Sheet

- Shubham Patel : 33%
- Harsh Murdeshwar : 34%
- Shubham Anand : 33%