CS335 Project - Milestone 2
Group 32

Shubham Patel	Harsh Murdeshwar	Shubham Anand
210709	210641	211020

April 3, 2024

1 Usage

We have used a wrapper script so that the process becomes easier. 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. The script name is `make_ast.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_ast.sh`.

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

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

2 Symbol Tables

A new symbol table is created whenever a new scope comes. Each symbol table consists of its entries. In the CSV, different symbol tables are separated by #####.

3 Parsing

We perform 3 parses.

3.1 Parse 1

In this parse, we fill in the symbol table. We have 3 types of symbol tables. One is the global symbol table. The second type is for functions. The third type is for classes. We record the line numbers at which entries have been declared. We also perform some error checking. In the symbol table, due to our multi-parse method, we have stored the entry's names.

3.2 Parse 2

In this parse, we perform type checking and store information about type-casting in the nodes of the parse tree. Our specification for valid types is as follows:
Definitions:

- Numeric - FLOAT, INT, BOOL
- Primitive - STR, FLOAT, INT, BOOL
- Precision of FLOAT > INT > BOOL
- For arithmetic operations - Numeric datatypes are allowed and the result is the maximum precision
- For int-specific math operations (like <<, >>) - Only INT datatypes are allowed
- For bool-specific operations (like &&, IF) - Primitive datatypes are allowed and they are type-casted to BOOL
- For Assignment Operation, Function Calls and Returns - Either the exact matching datatype is allowed or type-casting between INT< − >FLOAT and INT< − >BOOL is allowed

3.3 Parse 3

This parse is for computing the 3AC. The specifications of our implementation are as follows:

- int, float, and bool data types are stored explicitly on the function call stack, and their sizes are taken as 4 bytes, 4 bytes, and 1 byte, respectively
- For strings, the actual data of the string is stored in the read-only data segment and only the pointer to the string is stored on the stack

- Similarly, for lists and objects, the memory is allocated on the heap, and only the pointer to the start of the memory block is stored on the stack. Only the pointer is passed when assigning to or passing function arguments to list/class objects.
- Additionally, for lists, the first 4 bytes of the memory allocated for the list contains the number of elements present in the list

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