CS335 : Milestone 4

Arpit Kumar Rai (200190)
Avi Kumar (200229) Kunwar Preet Singh (200536)

April 2023

Tools and Utilities used

- Flex is used for our lexer, which is integrated with the parser. It returns tokens.
- Bison is used for implementing the parser.
- Make utility is used for automatic tracking for files and compilation.
- 3ac to x86 translator is implemented in python.
- A python script is used to provide automatic test runs.

Features implemented

- We have implemented all the basic features in the description.
- To enumerate these features :
  - Primitive data types
  - Multidimensional arrays (including 3D arrays)
  - All basic operators given in the description.
  - Control flow statements including if-else, for, while.
  - Support for method calling.
  - Support for recursion.
  - Support for System.out.println() for expressions.
  - Support for classes and objects.
- Syntax and Semantic checks have been implemented
- Type checking for every operator and datatype
- Interclass communication

Instructions for Compilation and Execution

- Note: End-to-end execution may be easily done without reading the following section. Instructions for end-to-end execution are present in the section "Instructions for Running test cases".

- Please change directory into *./milestone4/src/*
- Execute *make*
- Execute *./a.out - -input ./path/input.java - -output ./path/output* to give input and output to the parser that parses the java file and generates the corresponding 3ac. The symbol table is provided in *output_symtable.csv* while 3ac in *output_3ac.txt*.
- To generate the corresponding x86 code, place the 3ac file generated above in *./outputs* as *./outputs/proc.txt*. Note that the name of the file must be *proc.txt* and be placed in *./outputs* for further processing.
- Execute *python3 ../milestone4/to86.py > asm.s* to generate the assembly code.
- Compile assembly using *gcc asm.s* to generate the binary.
- Execute binary as *./a.out* to see the output.
- Note that - -input should be space separated from input path, as should be - -output from output path.
- Note that the implementation was heavily tested on gcc (GCC) 12.2.1 20221121 (Red Hat 12.2.1-4) and partially tested on gcc-9. Please use any of these versions (gcc-12 prefferably unless its not possible to use it on the tester's pc) to compile the assembly.

Command line options

- *- -input* : Can be used to set the input to the file that needs to be parsed. The path is expected to be space separated from *- -input.* Path can not be empty, it needs to be set.
- *- -output* : Can be used to set the output prefix. Any errors that may exist will be shown on the terminal via stderr. The path is expected to be space separated from *- -output.* Path can not be empty, it needs to be set. Two files with this prefix will be created, 3ac.txt, sym_table.csv.
- *- -verbose.* : On turning this on, verbose logs of parsing are provided.

Instructions for Running test cases

- We have provided 10 non-trival programs that may be compiled using the current compiler.
- We have provided an easy python script to automate the testing of these testcases.
- Please change directory into *./milestone4/*
- Execute *python3 compiler.py*
- The tests are named *./tests/test_[1-10].java*, corresponding outputs include *./outputs/[1-10]_symtable.csv*, *./outputs/[1-10]_3ac.txt*, *./outputs/asm[1-10].s* and *./outputs/asm[1-10].out*, .out files indicate the final binaries.

Other Information Regarding the Project

- While there were no significant extensions in 3ac from milestone 3 we did some minor changes in the syntax of 3ac to ease the process of translation into x86.
- No manual change is required to the assembly.
- All the basic features are supported.
- Contribution towards Project's Implementation :

Member Name	Roll Number	Member Email	Contribution (%)
Arpit Kumar Rai	200190	arpitkr20@iitk.ac.in	40 (%)
Kunwar Preet Singh	200536	kunwarps20@iitk.ac.in	40 (%)
Avi Kumar	200229	kumara20@iitk.ac.in	20 (%)