CS335 Project Milestone3

Pratham Jain (200712) , Gopal Aggarwal(200390) , Het Hitendrakumar Patel (200440)

April 2023

1 Tools and Utilities

- We have used flex for scanning the input file and it returns tokens.
- We have used bison for implementing the parser.We have generated grammar having no conflicts.
- Make utility is used to ease the process of writing multiple commands into one big command (i.e make).
- So, finally we have generated scanner and parser which takes as input an JAVA program and performs 2 functions. One of the parser creates Symbol table along with type checking and other parser generates 3ac codes with run-time environment.

2 Feature Implemented

- We have created type checking for various constructs and have created symbol table according to their scopes.
- Symbol table consists of Name, Type, Line number, Function Input Type, Function Output Type, Size and Scope.
- For representation of arrays, we have considered the type of element followed by number of brackets. For example, a 3D array of 'int', we have int[][][] as our type.
- For representation of arrays size, we have considered the lenght of array followed by cross($\times$). For example, a 3D array of 'int', we have int[][][] a = new int[5][7][2] as our size is 5X7X2.
- In case of class, the type name stored in symbol table is 'class' itself.
- In case of method, the type name stored in symbol table is 'method' itself.
- In case of constructor, the type name stored in symbol table is 'constructor' itself.

- In case of for loop, the type name stored in symbol table is 'for' itself. Similarly for other such cases we have done the same thing.
- We have used a feature that expands the data-type in order to preserve the information for example, a = 1 + 2.3, so here a is considered to be a 'float'.
- Apart from these features our parses produces 3ac codes for corresponding Java programs.
- We added run-time environment in 3ac code.
- We included space for actual parameters by pushing them just before procedure call by using pushparam in 3ac.
- We included space for old stack pointers to pop an activation by push %rbp them just before procedure call and pop %rbp after procedure return where %rbp is register which stores current the stack pointer and pop restores the old value in register.
- We included space for saved registers by push all necessary registers to be saved them just before procedure call and pop all them after procedure return where pop restores the old value in register .
- We included space for locals by push all local definitions by using pushlocal them in procedure run and pop all them after procedure return.
- We included space for return value by push return value in %rax register just before procedure end and assign that to caller's variable after procedure return.
- We have used vector of strings to store the 3ac codes in which each element of vector indicates one line or one statement of 3ac. In the end these codes are dumped into a text file where you can see the required output.

In this way , we have implemented all basic features as mandated. We have submitted 10 non trivial test programs which in all covers all constructs/aspects of JAVA language and compiles, parses and creates both the symbol table as well as generates the 3ac codes with run-time environments.

3 Command Line Options

- $--help$ or $--h$: This command helps the user running the program in knowing about various other commands required to run program or use any additional features.
  Examples of use

```
./parser -h
./parser -help
```

- $--$ *input* : This command is used for giving input to the program, i.e the path of the testfile (which is our input) is to be written after a space after this command. There should be a space between this command and inputfilename. Path can not be empty. Note that input file is of format .java
  Examples of use
  One of them -

```
./parser --input ..//tests/test_1.java --output test1.3ac
./parser --input ..//tests/test_5.java --output test5.3ac --verbose
```

  Here parser takes input from Java program and converts it into Symbol table stored in a .csv file and 3AC codes in a .3ac file.

- $--$ *output* : This command is used for producing output of the program, i.e. the file path(or name) of the output file where it should be produced after executing the parser. Path can not be empty. Note that output file is of format .3ac
  Examples of use
  One of them -

```
./parser --input ..//tests/test_1.java --output test1.3ac
./parser --input ..//tests/test_5.java --output test5.3ac --verbose
```

- $--$ *verbose* : The output of verbose is seen in the file *parser.out*. It shows if grammar has any conflicts and also has the actions and goto definitions for all states, it also has information about all the states. **The submitted parser.y has neither shift/shift nor any shift/reduce conflict**.
  Giving this command is optional
  You can give input, output , verbose command in any order

4 Instructions for Running Test Cases

- First go to directory *./milestone*3/*src/*
- Execute following commands :

```
> make clean
> make Java
```

  in succession

- Now write the command :

```
./parser  --input <inputfilepath> output <outputfilepath> --verbose
```

where $< inputfilepath >$ is file path of input file.

Note that input file is of format .java and output file can be of 2 types since we are having 2 different parsers giving 2 different kinds of output, one is giving a .csv file (which is the symbol table) and other is giving a .3ac file (which is the list of 3ac codes for the given program).

Also giving input filename is mandatory, by default the output file of tjewill be "symboltable.csv" file is generated in case of parser and ".3ac"