CS335A: Milestone 3

Submission By: Aarchie[200004] Harshit Kumar Tiwari[200432] Udit Prasad[201055]

1 Steps to run the program

Open the ‘milestone1/src‘ folder and type ‘make‘. It will show the different options which can be tried with makefile.

1. **help** to Show methods for each of the Makefile recipes
2. **build** to generate files for the executable. It will run the following commands-

   ```
   flex lexer.l
   bison -d -t parser.y
   g++ lex.yy.c parser.tab.c
   ```

3. **run** to run the executable using input like-

   ```
   make run input=file-path
   ```

   ‘or else run with terminal input. This would run-

   ```
   ./myASTgenerator input-file-path
   ```

   Also, output of ThreeAC file can be named optionally as using-

   ```
   run input=file-path.java output=output-file-path.3ac
   ```

   which runs like-

   ```
   ./myASTgenerator input-file-path dot-file-path output-file-path
   ```

   Meanwhile, dot and graph formed will also be stored in output-folder by default.

4. **t**o show Three Address Code

   ```
   make ThreeAC
   ```

5. **clean** to clean up the generated files

   ```
   make clean
   ```

   use this command before re-running test cases to clear all the output symbol Tables + three AC code + AST related files all in output folder.

2 Tools Used

flex: Used for Generating tokens from the input.
bison: Used for defining Syntax Grammar for the JAVA language.
Dot language: Used to generate the tree from the Grammatical syntax.
graphviz: Used to visualize the generated AST.
Makefile: Used to automate the working of the program for a specific input with different arguments such as -verbose,help,input,output
c++ programming language Used for the logical codeing portion of the program.

3 Three Address Code

3.1 Instructions:

- Assignment: It is simple assignment instruction. Ex- t0 = t1 + t2
- Unconditional jump: Jump to given label. Ex- goto L1
- Conditional jump: If condition is true then jump to given label . Ex- if cond goto L1
- Print: print the given variable Ex- print t0
- Param: To push the given parameter into the stack Ex- param t1
  - It is used in case of object instance creation using constructor where we will push the object reference.
  - It is used in case of method invocation when we pass some parameters.
  - It is used while creating array instance, where we will push the size of array to create memory.
- Popparam: To pop last pushed element from the stack. Ex- t1 = popparam . It is also used in similar fashion as previous one.
- call: To call the given function with n number of parameters. Ex- t5 := call func n
- getFromSymTable: It gets the offset of the variable from the given scope. Ex - getFromSymTable(scope,var). the resulting exact offset got is also printed for that scope after "=".
- pushArr: It is used to store the array elements in row-major order format.
  Ex- if we want to allocate the array 1,5,2,7, we will first allocate memory for the same and then push each of the elements.
  t10 := 16 – 4 elements * sizeof(int)
  param t10
  allocmem 1 – allocate memory
  t11 := popparam
  pushArr t11 1 0
  pushArr t11 5 4
  pushArr t11 2 8
  pushArr t11 7 12

3.2 Features Supported

The IR-Generator supports for the following JAVA language features-

- Primitive data types (e.g., int, long, float, double, and boolean)
- Multidimensional arrays supporting C-style declarations
- Basic operators:
  - Arithmetic operators
  - Preincrement, predecrement, postincrement, and postdecrement
  - Relational operators
  - Bitwise operators
  - Logical operators
  - Assignment operators
  - Ternary operator
- Control flow vila if-else, for, and while
- Methods and method calls
- Support for recursion
- the library function println()
- Support for classes and objects. For class definitions, supported public and private access modifiers
- Type casting
- Operator/Function disambiguation

3.3 Bonus Features

- Support for Strings, including a few operations like concatenation, support for printing with println().
- Support for Enhanced for loop.
- Array initializing using Array initializer.
- Declaring array using variables passed from parameter.

3.4 Assumptions

- `push basePointer` saves the current value of Base-Pointer on the stack and then moves it to the current Stack-pointer
- `pop basePointer` gets the old-basePointer from stack and stores it into basePointer to pop activation after returning.
- `basePointer := stackPointer` to move base pointer at current stackPointer position
- `return` instruction uses the old-BasePointer to reduce stack-pointer and deallocate space for local variables.
- `stackPointer` manipulation is used for allocating space for local variables and deallocating space for parameters.
- `push param` pushing parameter itself allocates space in stack and saves the values on stack.
- Constructors need to explicitly made before creating an instance.

4 Symbol Tables

- We have defined our Global Symbol table named as **global_sym_table** in the file **parser.y**.
- In the folder **output/symTables/**, all the Symbol Tables of corresponding scopes are printed in csv file with its name as -
  - for class : className.csv
  - for methods: className_methodName.csv
  - for constructors: cons_className
- For inherited classes, all the contents of parent class's scope's symbol table is copied.