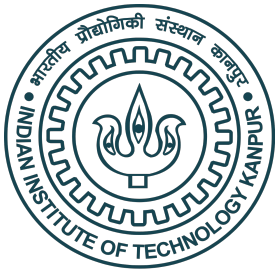

CS335 Compiler

MileStone 2

Swarnendu Biswas

March 23, 2023

Jaya Gupta (200471)

Harshit Bansal (200428)

Mohit Gupta (200597)

---

1 Requirements

- g++ $\geq$ 11, gcc $\geq$ 11 : The system should also have g++ installed.
- flex: The environment should have flex installed.
  - If its Linux then it can be installed from the command

```
sudo apt install flex
```

- Bison : The environment should have bison installed.
  - If its Linux then it can be installed from the command

```
sudo apt install bison
```

2 Execution Instructions

Compilation and Execution Instructions

```
make
./build/milestone/javap --input <input-file-name> --output
   <output-file-name>
dot -Tpng <dot_script> > <output_png_file>
```

To parse all the testcases in tests folder, execute run.sh script.

Command Line Options

```
Usage: javap [-h] --input VAR --output VAR [--verbose]

Optional arguments:
  -h, --help    shows help message and exits
```

```
-v, --version prints version information and exits
--input       java file to parse [required]
--output      output dot file [required]
--verbose     increase output verbosity for parser
```

3 Symbol Table Implementation

- We have created a global symbol table which is a map of ¡class_name¿ and Class_Symbol_Table.

```
unordered_map<string, ClassDefinition *> classes;
```

- Each Class_Symbol_Table consists information about its instance variable and a map of its Methods_Symbol_Table and Constructors_Symbol_Table.

```
int modifiers;
string name;
Type* type;
unordered_map<string, SymTabEntry *> inst_vars;
unordered_map<string, deque< MethodDefinition *> &> methods;
vector< MethodDefinition *> constructors;
```

- Each Function_Symbol_Table has information about its argument types, return types and the local symbols information.

```
unordered_map<string, SymTabEntry *> sym_table;//local symbol
                                                //information
string method_name;  // name of function
int8_t modifier;
Type *return_type; // return Type information
ClassDefinition *container_class;
vector<Type *> args;
```

- For scope management inside a function(like if-else, while, for), we have used symbol table of the parent method only and cleared the scope information from the symbol table of (if-else, while, for) when we exit the scope. The information is dumped in a file when a scope is cleared.

4 Type Checking

- We have run many experiments on real Java compilers to understand type checking rules to make sure our type checking rules agree with standards.
- We have done it for all expressions and operators, methods, classes. Even static, public, private and final type checking as specified on the forum are implemented.

- We have added appropriate error messages with the line numbers also. System,

- We have also implemented type-casting as asked in the description of the milestone 2.

5 Three - AC Code

- Labels are denoted by

```
LabelName :
```

- The 3AC code is written in the form

```
operator argument1 argument2 result
```

- For less number of arguments it leaves that argument empty
- Array initialization instruction is directly done as variables.
- Function/constructor initialization are done as follows :

```
popparam parametername
/* Function Statements */
return tempvar
```

- Function call are done as follows :

```
call func_name argument_count temporary_variable
//temporary variable stores return value
```

- Eg :

```
int a[][] = new int[2][3];
x = Func(y)
A c = new A(b);
```

- The above code will give 3ac as:

```
= new int[2][3] t0
= t0 a[][]
param y
call Func 1 t1
```

```
        = t1 x
        param b
        newcall A 1 t2
        = t2 c
```

6 Additional Features Added

- Function Polymorphism : Our code supports polymorphism for functions and class constructors.

```
void func1(long a) {
    // Function 1 with name func1
}
static int func1(int a) {
    // Function 2 with same name func1 but with different
   argument list
}
```

Our code is able to handle above cases.

- Type Casting: Our code supports typeCasting

```
    int var1 = 2;
    char var2 = 'c';

    long var3 = (long) var1;
    long var4 = (long) var2;
```

Our code is able to handle above cases.

- Constructors: Support for multiple class constructors is provided. Also according to Java17 rules, if any of the constructor is defined by the user, then the default constructor should be compusarily defined is also checked.

```
    class Me {
        int ins1;
        int ins2;
        boolean ins3;
        Me(int a, int b) {
            ins1 = a;
            ins2 = b;
        }

        Me(boolean a){
            ins3 = true;
        }
```

```
        Me () {
            // default constructor
        }
    }

    // This code will work correctly.
```

```
    class Me {
        int ins1;
        int ins2;
        boolean ins3;
        Me(int a, int b) {
            ins1 = a;
            ins2 = b;
        }

        Me(boolean a){
            ins3 = true;
        }
    }

    // This code will give error as default constructor is not
   defined.
```

- Class Instance Initialization: Initialisation with new keyword is allowed.
- Do While loop is supported.

7 Assumptions

- Array Initialization in curly braces $\{2,4,5\}$ is not supported. Array can only be initialized by new keyword.
- System.out.println and Strings are not supported. String Type is not supported.
- Scope is kept as a number. It is incremented when a new scope(function, for, if-else, while, do while) is seen and decremented as we exit from these scopes. This is done for ease of usage and does not affect the performance or accuracy of parser and semantic analyser in any case.
- Each variable scope is assigned a number. So in cases like below:

```
if(true){
    int x;
} else {
    int x;
}
```

```
// Here in symbol table dump, there will be two similar entries
   as both x have same name, modifiers, type and same scope(as
   scope is a number). Both if and else have same scope number
   as per the policy in point above.
```

- We have throwing errors for the basic type-checking errors which are the static compilation checks done while parsing the grammar.
- We have added support to access modifiers like public, private, static and final only.
- All the basic features as mentioned have been incorporated in the submission.
- In three AC, we have given class objects access (obj.a) in code as obj.a only in 3AC dump. This was because of the confusion, if there are circular object declaration in 2 classes like

```
class Class1 {
    Class2 ins;
}

class Class2 {
    Class1 ins;
}
// In this case the offset of each of the classes can't be
   calculated due to recursion.
```

- Our 3AC looks slightly different from the 3AC example of piazza due to some notational convention adopted by us as described in the section Three-AC Code above.