# CS335A: Compiler Design

Milestone 3

*Harshit Raj* · 200433
*Kumar Arpit* · 200532
*Akshan Agrawal* · 180061

April 4, 2023

---

## 1 Dependencies

- python 3
- pip
- Tools used
  - Graphviz
  - ply (python-lex-yacc)
  - argparse

## 2 How to Run

```
cd $DIR/milestone3
pip install -r requirements.txt
python3 src/parse.py --input /path/to/file.java [--output
 ↪  /path/to/output] [--all] [--verbose]
dot -Tps src/output -o src/output.ps
xdg-open src/output.ps
```

PS: Verbose will pretty print the symbol table and TAC.
Alternatively, you can also use the Makefile

```
    make dev ARG=x
```

this will run `tests/test_x.java`

## 3 Command Line Arguments:

- input: Specify Input file name
- output: Specify output file, defaults to *javao*
- all: Generates parse tree in dot file instead of AST
- verbose: Prints symbol table and 3ac to the terminal
- help: Shows help

## 4 Features Implemented

- All basic required features as specified.
- Do while statement.
- Support for Strings.
- Constructors.
- Type Casting.

## 5 Relevant Code Files

- lexer.py: Responsible for lexer specification or lexer definition, contains instructions for a lexer program to recognize and categorize the lexical elements or tokens in a source code file.
- parse.py: Responsible for parser specification or parser definition, contains instructions for a parser program to analyze the structure of a source code file based on the sequence of tokens produced by the lexer program.
- dot.py: Responsible for representation of the structure of a program's source code in the form of a parse tree or syntax tree.
- symbol_table.py: Responsible for providing template for symbol table (data structure) for keeping track of various symbols.
- symbol_tac.py: Responsible for generation of symbol table with the help of DFS traversal of the parse tree governed under the template supported by symbol_table.py file.
- tac.py: Responsible for generation of 3AC code representation of the program.
- utils.py: Responsible for providing certain helper functions (built by us) to the symbol_tac.py file.

## 6 CSV Header Format

- class: className, symbolType(class), symbolTableName, modifiers, parentClass, interfaces, size
- method: methodName, methodSignature, symbolType(method), symbolTableName, methodParameters, returnType, modifiers, throwsList
- variable: variableName, symbolType(variable), dataType, size, offset, modifiers, numberOfDimensions, arrayOfDimensions
- block: blockName, symbolType(block), symbolTableName

# 7 Memory Management

In Milestone 3, we added memory management allocated memory to local variables, parameters, etc.

## 7.1 Stack management

The stack is considered infinite. And all primitives whose size is known are kept on the stack. All those unknowns are stored as pointers to heap memory (8 bytes for the pointer).

### 7.1.1 Stack pointer management

The stack pointer is incremented on variable/field declarators.

### 7.1.2 Base pointer management

The base pointer is updated when entering a function or leaving a function.

## 7.2 Heap Management

The heap is considered automatically managed, not managed by our compiler. We are using `memalloc <size>` to allocate `size` bytes on the heap and return the starting address.