Compiler Design

> Abhinav Kumar Singh(200017)
> Harshit Gupta(200429)
> Ronit Mittal(200820)

Executing the code

We have used the following tools:

- **Bison**
- **Flex**
- **C++ - Makefile**
- **Dot - Graphviz**
- **Nasm**

The **file structure** is as follows:

```
.
├── doc
│   └── cs335_project_report.pdf
├── Makefile
├── src
│   ├── 3ac.cpp
│   ├── 3ac.h
│   ├── 3acstack.cpp
│   ├── 3acstack.h
│   ├── ast.cpp
│   ├── ast.h
│   ├── codegen.cpp
│   ├── codegen.h
│   ├── commandline.cpp
│   ├── commandline.h
│   ├── globalfuncvars.h
│   ├── lexer.l
│   ├── main.cpp
│   ├── parser.ypp
│   ├── symboltable.cpp
│   ├── symboltable.h
│   ├── typecheck.cpp
│   └── typecheck.h
```

```
├── sym_table
└── tests
    ├── test_10.java
    ├── test_1.java
    ├── test_2.java
    ├── test_3.java
    ├── test_4.java
    ├── test_5.java
    ├── test_6.java
    ├── test_7.java
    ├── test_8.java
    └── test_9.java
```

To **compile the code**, run the following command(in folder `milestone4`):

```
make run
```

The make file generates a binary, `run` which can then be executed as:

```
./run -i <input_file.java> -o <output_file.asm>
```

The list of commands for the binary are:

```
Usage: ./run -i <input file> -o <output file>
[REQUIRED]-i or --input <input file>: input file name
[REQUIRED]-o or --output <output file>: output file name
[OPTIONAL]-v or --verbose: verbose mode
[OPTIONAL]-a or --ast <DOT output file name>: print ast
[OPTIONAL]-s or --sym <symbol table output folder>: print symbol table
```

The `--sym` option requires you to create a folder with the name given as input. All the tables are automatically stored in it in `.csv` format.

The `3ac` is generated in the output file `-o`.

To compile and run any test case, you can use:

```
make tc<x>
```

Where `x` is the test case number.

Functionalities

- The compiler is capable of compiling Java code with datatype `int`. It can do complex arithmetic.
- It also supports function calls up to a **depth of 1**. (there is no support for recursion)
- We have created a print function, `printRAX`, which is capable of printing integer variables only.
- The compiler supports `if`, `else`, `do-while` and `for` statements and can be nested.

Limitations

- Each variable should have a unique name.
  - Could have been fixed while creating the symbol table.
  - **We could have given an alias to each variable, and each unique variable in that scope could have had a different alias.**
- Callee functions should be defined below caller functions.
  - Mainly because memory allocation is static.
  - Function Parameters are assigned stack locations in the calling function and are then used in the callee function.
  - **We could have generated each 3ac separately and then reordered to overcome this.**
- Each function can be called only once.
  - **This is a debugging issue**. For some reason, the return address is overwritten when multiple calls are done for the same function.
  - We could not support recursion for the same reason.
- No support for register spilling
  - The program shows an error when more than the available number of registers are in use.
  - We could have generated an algorithm to store registers in memory in such cases. What we thought of was using the register, which will be used after the most number of lines of code.
- We could not support any other basic functionalities due to time constraints.

Contribution done by Team Members

Name	Roll No	Email	Contribution
Abhinav Kumar Singh	200017	abhinavk20@iitk.ac.in	50
Harshit Gupta	200429	guptah20@iitk.ac.in	25
Ronit Mittal	200820	ronitm20@iitk.ac.in	25