PA5 -- Three Address Code Generation

---

1. Description

For this assignment you will implement the logic to generate the three address code(TAC) for the input source code. The TAC will later be used for optimization and MIPS generation in the later assignments.

The approach to implement TAC generation is similar to the approach you used in PA4 when you are implementing the logic for semantic check. Instead of implementing Check() method for ast classes, you will implement **Emit()** for the ast classes. The general workflow of the program should be the same as PA3: The TAC generation will be fired off by calling **Program->Emit()** in parser.y**,** and then the Emit() will be called on every node of the AST, and the corresponding TAC will be generated.

2. Starter Code

Makefile	used to build the parser
utility.h / .cc	interface / implementation of various utility functions
errors.h / .cc	class that defines error reporting methods
location.h	class that defines type for token location
lexer.h	type definitions and prototype declarations for lexer
lexer.l	flex input file(use your own from PA3)
parser.h	declare parser functions and types
parser.y	implementation of the parser(use your own from PA3)
main.cc	the entry point where the whole program will be started
samples	directory contains test input files
ast.h / .cc	**interface / implemention of AST node classes**
ast_type.h / .cc	**interface / implemention of AST type classes**
ast_decl.h / .cc	**interface / implemention of AST declaration classes**
ast_expr.h / .cc	**interface / implemention of AST expression classes**
ast_stmt.h / .cc	**interface / implemention of AST statement classes**
symtable.h / .cc	interface / implemention of symbol table

When you are going through the starter code, please read the comment as well. The comments are really helpful in explaining the structure of the code.

In this assignment you need to implement Emit() methods for most of the ast classes so you will need to modify all the ast classes files.

We will provide sample input and output for all kinds of expressions and statements so that you can take a look at them and get familiar with what does the output TAC look like with the corresponding input source code. You can find them in **samples** folder.

3. Example

Before you do a make and run some sample test cases, make sure you change the code in your **parser.y;**

Your old parser.y from PA4 should have the code `program->Check()`, you should change it to `program->Emit()` for PA5.

Then you can make and try to run some sample test cases. The following three test cases will pass by running the starter code:

./parser < samples/**test1.java**
./parser < samples/**arithmetic_1.java**
./parser < samples/**arithmetic_2.java**

The following classes' Emit() methods have already been **completely** implemented, so you do NOT need to modify them unless any unexpected issue came up and we will announce the change on piazza:

```
string IntConstant::Emit()
string BoolConstant::Emit()
string Operator::Emit()
string StmtBlock::Emit()
```

The following classes' Emit() methods have been **partially** implemented, you need to change or complete the implementation of the following Emit() methods.

```
string Program::Emit()
string VarDecl::Emit()
string VarExpr::Emit()
string ArithmeticExpr::Emit()
```

Recommended steps to start:

1. Look at **test1.java / .out, arithmetic_1.java / .out** and **arithmetic_2.java / .out**. Get familiar with the conversion from source code to TAC.
2. Look at the given implementation of Emit() method, figure out how do they help make the above 3 test cases pass.
3. Look at other test cases and output and implement other Emit() methods.

4. Testing your work

We will provide some simple test cases and the corresponding output which you can find in the **samples** folder. You should definitely come up with your own test cases as well to test that you have covered all the potential semantic errors.

5. Grading

You can go to any teaching staff's lab hours and ask for a check-off when you are done with your assignment. You will be asked to run your assignment against the given test cases, and we will check the output of your assignment. You also need to generally describe how did you approach the assignment to the teaching staff.