COP-3402 Systems Software
09/17 Tue

Reminder on Hw3: write corresponding LLVM IR code.

Please refer to the diagram under syllabus/projects. Compiler takes simplec program program.simplec as input. Your project is to write compiler to convert simplec into .ll program. And clang it to have executable. *fgetc* each character, turn them into tokens and process them.

Project 0:
One function per token
One function for statement
Save lexemes
Emit LLVM IR, filling in the template

print 42 + 31; simplec program (input)
your compiler reads each character at a time and recognizes there is a complete token in it, print, numbers, operation, semicolon, etc.

```
// true if recognized, false otherwise
 printT(){
 c = fgetc(file);
 // method1 read in a sequence of alphabetical characters
 // malloc a buffer
 while isalpha
   add each c to the buffer
   c = fgetc(file);
 strncmp to check for print

// method2 read one character at a time and check
 if (‘p’ == c){
}
c = fgetc(file);
if (‘r’ ==c){
}
c = fgetc(file);
etc..
 else {
// do error handling
     return false;
//

// check for EOF
```

```
if (EOF == c)

// lexeme string or null
numberT(){
c = fgetc(file);
// is it a minus sign?
isdigit()
ungetc(c, file)
}

//function for each token

//whitespace


statement(){
// at the beginning of file
printT();
//check if recognized
OP1 = numberT();
c = fgetc(file);

if (c beginning of number) {
oplexeme = operatorT();
//convert from simplec operator to LLVM operator, e.g., '+' "add nsw"
OP2 = numberT();
semicolonT();
"\%t%d = %s i32 %s, %s", gettempvar(), operatorstring, op1, op2
// VARNAME = OPNAME i32 OP1, OP2
//emit LLVM instruction
//emit the call to print
}
If (c a semicolon) {
semicolonT();
}


#define NUMBER 1
#define PRINT 2
…

struct token {
  char *lexeme;
```

```
  int tokenid;
};

//takes a file returns an array of tokens
}
```

Referred briefly back to the Regular Expressions.
epsilon means the empty string.
We will write code that matches the pattern.
IDENTIFIER LETTER (LETTER|DIGIT)*

Finite Automata:
Called Finite state machine or finite state automata
Defines finite sets of States with initial state and transitions between them
Examples: vending machines, traffic lights
turnstile states: locked and unlocked
turnstile transitions: push and coin
Transitions move from state to state
A state is actually an instruction pointer.

Nondeterministic FA
Multiple states allowed, same symbol, multiple transitions, epsilon transitions

Deterministic FA
One state at a time

Any regular expression can be represented with an FA
Any NFA can be represented with a DFA

Ex. NonD FA:
(a|b)*abb

State-Transition table

	a	**b**	**epsilon**
1	{1,2}	1	x
2	x	3	x
3	x	4	x
4	x	4	x