COP-3402 Systems Software
09/10 Tue

Alternatives to Vagrant for Windows and Linux users:

For Windows users, using WSL with Ubuntu 18 should be sufficient instead of using Vagrant. Make sure you've got clang v6 (clang --version to find out), git, and make installed.

For Linux users, if you are on Ubuntu 18 already and have clang v6, then you should be fine.

Compiler Overview:
The reason C runs is because it is translated by the compiler to be run in the hardware.

The diagram of a translation of an assignment statement(from the book):
position = initial + rate * 60 => lexical analyzer => syntax analyzer => semantic analyzer => int. code generator => code optimizer

Processing the input file:
1. Open the input file by name
2. Read one character at a time
3. Loop until you obtain the end of input file
4. Checking by Helper functions
5. Using a char buffer

Generating output file:
1. Read characters until seeing a full statement
2. Print the LLVM IR with printf()
3. Use format strings

Each project is a complete compiler including:
1. Arit. Operations
2. Arit. Expressions
3. Variables
4. Control-flow structures
5. Functions

And will cover:
1. Target language
2. Source language
3. Compiler theory and algorithms

Proj0 overview/syntax:
program
= statement*

statement
= PRINT expression SEMI

expression

= NUMBER PLUS NUMBER
…

Ex:
print 5;
print -5 - 7;

-5 Recognized as a minus sign and number five; so minus 5. The other minus is recognized as the subtraction operation. If there is no number before minus, or if minus comes after print statement, or there is another operator (i.e., + etc.) before minus, then it is a sign.

A print statement is composed of the print keyword and an expression followed by a semicolon.

Architecture Suggestion for Projects:
Create a function for each token, symbol, unit, expression, statement, etc.

For projects;
Work on the template, which is under syllabus/projects.

Integer constants:

1. i32 -43
2. i32 means a 32-bit signed integer
3. -43 is the decimal
4. We will only use i32 values for our compiler.

Variable names; e.g., %t1
Arithmetic Instructions:
Assignments; (unlimited number of registers)
%t1 = add nsw i32 2, 3
print 2+3;
i32 identifies both the operands.
Variables can only be assigned once.
Generate variables by incrementing a counter.
We need temporary variables (i.e., %t1, etc.) in order to store them.
Other instructions: sub nsw, mul nsw, sdiv, srem
If it is sub, the order matters; 2 - 3

Printing:
template.ll has a print_integer function:
Calls printf and clang links with the C lib.
Use a function with the call instruction:
call void @print_integer(i32 %t1)

HOST VM
vagrant ssh => bash
syllabus/projects => /vagrant

cd /vagrant

cd examples
ls
ls all.ll
clang -o all all.ll
./all

Ex:
%t5 = srem i32 10, 3
call void @print_integer(i32 %t5)
print 10 % 3;
For constant:
call void @print_integer(i32 2)
print 2;

Lexing:
A lexer groups characters into words
Source files are text files
Compilers use algorithms to recognize words

Char type
char c = ‘a’;
c = fgetc(file);

Lexer reads and buffers characters:
Reads each character from a file
Buffer them into an array
Complete the word and go next
Command:
hexyl helloworld.c

Demo: Compiling LLVM IR
cd /vagrant
cd examples
cat helloworld.ll
=> We can use clang to turn these llvm files into machine code that can be executed.
clang -o helloworld helloworld.ll
=> Now we have our helloworld program:
ls -latrh helloworld
./helloworld
=> prints hello, world
clang -o all all.ll
./all
5
-1
6
3
1
2