Administrative

• Most things are on the class web site:
  • http://www.cs.ucdavis.edu/~peisert/teaching/ecs142-sp09/
  
• Grades and discussions are on SmartSite (you should have access)
Staff

• Instructor: Dr. Sean Peisert
• Email: peisert@cs.ucdavis.edu
• Office: Watershed 2111 (map on web)
• TA: Daryl Posnett
• Email: dpposnett@ucdavis.edu
Dates & Times

- **Lecture:**
  - Bainer Hall 1130, MWF 9a-9:50a

- **Discussion Section**
  - Olson Hall 217, M 4:10p-5p

- **Midterm:** TBA (probably May 1 in class)

- **Final Exam:** Thursday, June 11, 1:00pm
Text & References

- The first edition is fine too.
- More is coming on the web page (e.g., SVN reference, makefile reference)
Course Goals

• Learn about compilers
• Use software engineering
• Build a working, complete compiler in five steps.
Grading

• The grade for ECS 142 will be based on five projects, one midterm, and a final, as follows:
  • Project 1 (lexer): 5% of final grade
  • Project 2 (parser): 15% of final grade
  • Project 3 (semcheck): 15% of final grade
  • Project 4 (semcheck, part II): 15% of final grade
  • Project 5 (codegen/optimization): 15% of final grade
  • Midterm: 10% of final grade
  • Final: 25% of final grade
The Class

- Not hard, but time consuming.
- Requires extensive programming.
- Lecture will contain both theory and program information.
- Discussion section will clarify the project.
- Midterm will be in class.
- Not a good quarter to take 20 units.
The Compiler

• Five projects

• Lexer, Parser, Semantic Checker (Parts 1 & 2), Code Generator

• All must be turned in, regardless of your exam scores, for a passing grade.

• Working with a partner is highly recommended, but not required.
Grading Standards

- All assignments must be turned in to pass the course.
- Just turning in all of the assignments and taking the exams doesn’t warrant an A or even a B.
Programming

• Java and C++ are both fine (*no* Perl/Python).

• You’re welcome to use existing data structures (e.g., linked lists, stacks, n-child trees). STL for C++ is okay.

• CVS or SVN is highly recommended for saving code, especially when you have a partner.

• Work ahead and finish before the deadline.
Academic Integrity

• You can work with one partner.
• Don’t show anyone other than your partner your code.
• You cannot reuse either your own or someone else’s existing code except for basic data structures (linked lists, stacks, and trees).
• We use moss to compare projects (even from past courses). Every quarter, there are people caught cheating.
• Don’t test us.
Projects

- Projects are 65% of the course.
- Don’t start the projects the weekend before they are due.
- Apply good software engineering principles.
- Divide the project among yourself and your partner.
Grading Projects

- We’ll compare the output of your compiler to the output of ours.
- We’ll supply minimal test cases (you should write some of your own, too).
- Style/organization counts in the margins.
Compiler History

- Compilers (do pre-processing)
- Interpreters (don’t)
- Just-in-Time Compilation
Our Compiler

• Compiling a language called “Cool” Your compiler will turn Cool into MIPS assembly.

• An assembler will translate the assembly language into machine code and then into the final executable.
Lexing, parsing, and semantic checking can be understood by analogy to understanding how humans comprehend English.
Lexical Analysis

• First step: recognize words

This is a sentence.

Note:

• Capital “T” (start of sentence)
• Blank “ ” (word sep)
• Period “.” (end of sentence)
Lexical Analysis

- Lexical analyzer divides program text into "words" or "tokens"

  If \( x == y \) then \( z = 1 \); else \( z = 2 \);

- Units:

  if, \( x, ==, y \), then, \( z, =, 1 \), ;, else, \( z, =, 2 \), ;
Sample Program

• Given this program:

```java
import "something"
Class /* Comment */ A extends B {
    void smallproc (Integer xyz) {
    }
    public void main static (String# args) {
        int args = xyz;
        int x = 1.2;
        System.out.println ("Hello world" + 23);
    }
}
```
• Groups input into “token” units:
  • ‘I’ ‘M’ ‘P’ ‘O’ ‘R’ ‘T’ = IMPORT keyword (T_IMPORT)
  • ‘I’ ‘.’ ‘2’ = 1.2 real literal (T_REAL)
  • ‘x’ ‘y’ ‘z’ = Identifier xyz (T_ID)
  • All of these are tokens
  • Gives errors, too! (# not a valid character in Java)
Parser

- Matches the tokens to a language syntax specification.
- Knows that:
  
  `public void main static`

- ...doesn’t match any rule in Java and reports the error.
- What is syntax? The grammatical structure of the language. Like sentence diagramming.
Sentence Diagramming Example
Program Diagramming Example
Semantic Checker

• Looks for “meaning.” Meaning is hard; semantic correctness is a more limited but feasible check

• Part 1: Scoping and visibility.

• Part 2: Type-checking:

```
String x, y;

int z;

...

z = x + y;
```

• Is this semantically correct?
Semantic Analysis in English

Jack said Jerry left his assignment at home.

- What does “his” refer to?

Jack said Jack left his assignment at home.

- How many “Jacks” are there?
Semantic Analysis in Programming

```c
{  
    int Jack = 3;
    {
        int Jack = 4;  // this defn is used in C++
        cout << Jack;
    }
}
```
Semantic Analysis in Programming

Jack left her homework at home.

- Checks type (e.g., gender).
Code Generation

- Output MIPS Assembly Code
- \( z = x + y \) becomes assembly code that loads \( x \) and \( y \) from memory, adds them, and stores them back in the proper memory location for \( z \).
- Requires proper syntax and semantic correctness and a symbol table that tells us where to look in memory.
Optimization

- Most optimization covered in graduate classes, but we will do a very small amount.
- \( X = Y \times 0 \) vs. \( X = 0 \)
Symbol Table

• Database which keeps track of variables.
• Type
• Scoping
• Memory location
  • int x; // stores x into symtable as integer
  • ...
  • x=23; // looks up x and verifies it is an integer
Overview
Homework

• Optional but recommended: you are encouraged to find a partner.

• Contact me if you don’t have a CSIF account.

• Reading (Dragon book)
  • Skim ch. 1 by Wednesday (4/1).

• Project 1 (Lexer) due April 10, 11:55pm. This will be assigned on Wednesday (4/1).