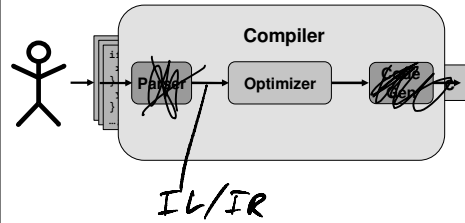


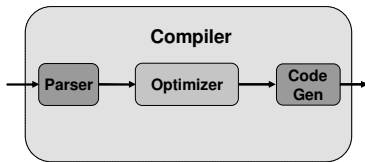
~~Advanced Optimizers~~

CSE 231  
Instructor: Sorin Lerner

Let's look at a compiler



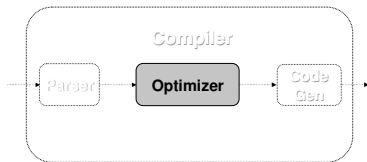
Let's look at a compiler



~~Advanced Optimizers~~

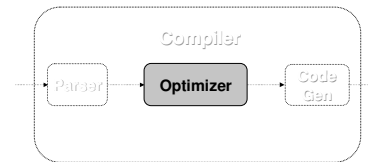
CSE 231  
Instructor: Sorin Lerner

What does an optimizer do?



1. Analyze input prog
2. Make it more efficient

What does an optimizer do?



1. Compute information about a program
2. Use that information to perform program transformations  
(with the goal of improving some metric, e.g. performance)

### What do these tools have in common?

- Bug finders
- Program verifiers
- Code refactoring tools
- Garbage collectors
- Runtime monitoring system
- And... compilers

### What do these tools have in common?

- Bug finders
- Program verifiers
- Code refactoring tools
- Garbage collectors
- Runtime monitoring system
- And... compilers

They all analyze and transform programs  
We will learn about the techniques underlying all  
these tools

## Program Analyses, Transformations, and Applications

CSE 231  
Instructor: Sorin Lerner

### Course goals

- Understand basic techniques for doing program analyses and transformations
  - these techniques are the cornerstone of a variety of program analysis tools
  - they may come in handy, no matter what research you end up doing
- Get a feeling for what research is like in the area by reading research papers, and getting your feet wet in a small research project
  - useful if you don't have a research area picked
  - also useful if you have a research area picked: seeing what research is like in other fields will give you a broader perspective

### Course topics

- Techniques for representing programs
- Techniques for analyzing and transforming programs
- Applications of these techniques

### Course topics (more details)

- Representations
  - Abstract Syntax Tree
  - Control Flow Graph
  - Dataflow Graph
  - Static Single Assignment
  - Control Dependence Graph
  - Program Dependence Graph

## Course topics (more details)

- Analysis/Transformation Algorithms  $x := 5$ 
  - Dataflow Analysis
  - Interprocedural analysis  $*p := \dots$
  - Pointer analysis  $y := x + 3$
  - Abstract interpretation
  - Rule-based analyses and transformations
  - Constraint-based analysis
  - Interaction between transformations and analyses
  - Maintaining the program representation intact

## Course topics (more details)

- Applications
  - Scalar optimizations
  - Loop optimizations
  - Object oriented optimizations
  - Program verification
  - Bug finding
  - Garbage collection

## Course pre-requisites

- No compilers background necessary
- Some familiarity with lattices
  - I will review what is necessary in class, but it helps if you know it already
- Some familiarity with functional programming and object-oriented programming
  - we will look at optimization techniques for these kinds of languages
- A standard undergrad cs curriculum will most likely cover the above
  - Talk to me if you think you don't have the pre-requisites

## Course work

- Assignments (20%)
  - about 2 assignments throughout the quarter
- Course Readings (5%)
  - selected readings from the literature
  - write short paper reviews
- Participation in class (5%)
- Take-home final (20%)
- Course project (50%)

## Course project

- Goal of the project
  - Get some hands on experience with compilers
  - Get a feel for what research is like in PL
- Two kinds of projects:
  - research-y: explore some interesting ideas and try them out. Depending on the project, you may or may not need an implementation
  - implementation-y: pick some existing idea out there, and implement it (you'll be amazed what that does to your understanding of said idea...)

## Course project

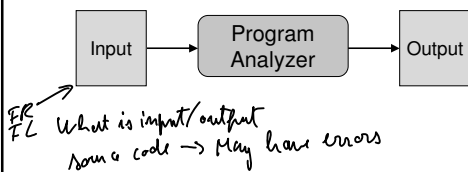
- Groups of 2-3 (1 ok if you convince me of it)
- I STRONGLY encourage you to pick something that (is related to)/(will advance) your research outside of this class
- Milestones
  - Project proposal (due 2 weeks in)
  - Mid-point presentation (due 5 weeks in)
  - Final presentation and written report (due end of quarter)

## Administrative info

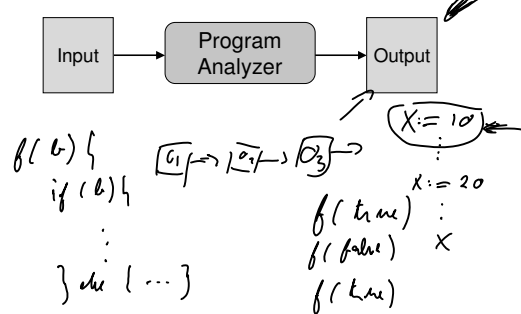
- Class web page is up
  - <http://www.cse.ucsd.edu/classes/fa07/cse231/>
- Will post lectures, assignments, etc.
- Subscribe to the class mailing list
  - instructions are on the class web page
  - click on "course syllabus"

## Questions?

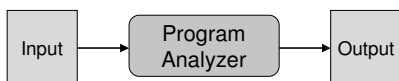
## Program Analyzer Issues (discuss)



## Program Analyzer Issues (discuss)



## Program Analyzer Issues (discuss)




## Input issues

Instructor's discussion notes



- Input is a program, but...
- What language is the program written in?
  - imperative vs. functional vs. object-oriented? maybe even declarative?
  - what pointer model does the language use?
  - reflection, exceptions, continuations?
  - type system trusted or not?
  - one often analyzes an intermediate language... how does one design such a language?

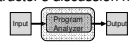
Instructor's discussion notes



## Input issues

- How much of the program do we see?
  - all?
  - one file at a time?
  - one library at a time?
  - reflection...
- Any additional inputs?
  - any human help?
  - profile info?


Instructor's discussion notes



## Analysis issues

- Analysis/compilation model
  - Separate compilation/analysis
    - quick, but no opportunities for interprocedural analysis
  - Link-time
    - allows interprocedural and whole program analysis
    - but what about shared precompiled libraries?
    - and what about compile-time?
  - Run-time
    - best optimization/analysis potential (can even use run-time state as additional information)
    - can handle run-time extensions to the program
    - but severe pressure to limit compilation time
  - Selective run-time compilation
    - choose what part of compilation to delay until run-time
    - can balance compile-time/benefit tradeoffs

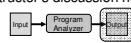
Instructor's discussion notes



## Analysis issues

- Does running-time matter?
  - for use in IDE?
  - or in overnight compile?

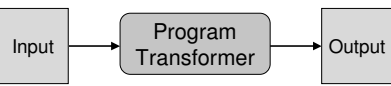
Instructor's discussion notes




## Output issues

- Form of output varies widely, depending on analysis
  - alias information
  - constantness information
  - loop terminates/does not terminate
- Correctness of analysis results
  - depends on what the results are used for
  - are we attempting to design algorithm for solving undecidable problems?
  - notion of approximation
  - statistical output

## Program Transformation Issues (discuss)

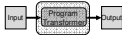


Instructor's discussion notes



## Input issues

- A program, and ...
- Program analysis results
- Profile info?
- Anything else?



## Transformation issues

- What is profitable?
- What order to perform transformations?
- What happens to the program representation?
- What happens to the computed information? For example alias information? Need to recompute?



## Output issues

- Output in same IL as input?
- Should the output program behave the same way as the input program?