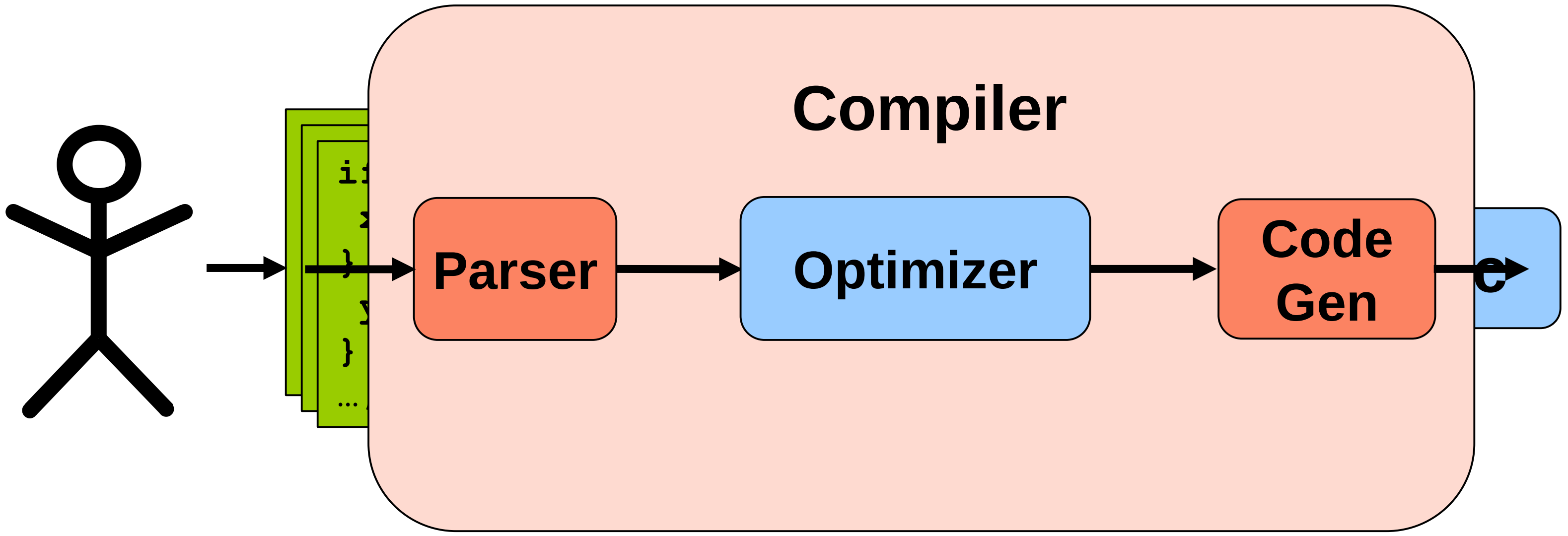
# Advanced Compiler Design

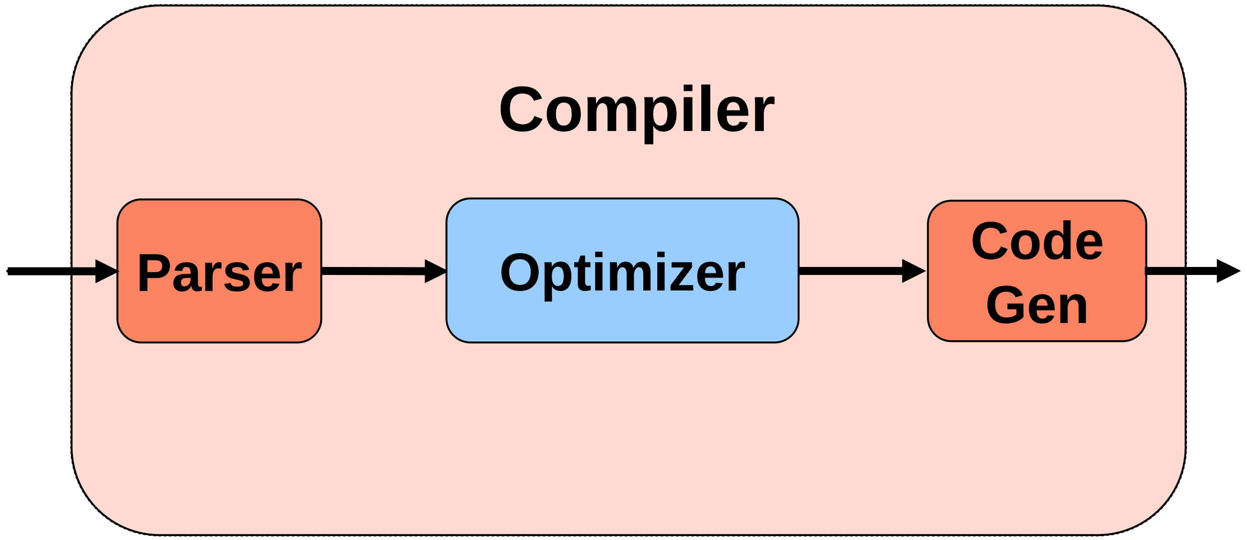
CSE 231

Instructor: Sorin Lerner

# Let’s look at a compiler



## Let’s look at a compiler

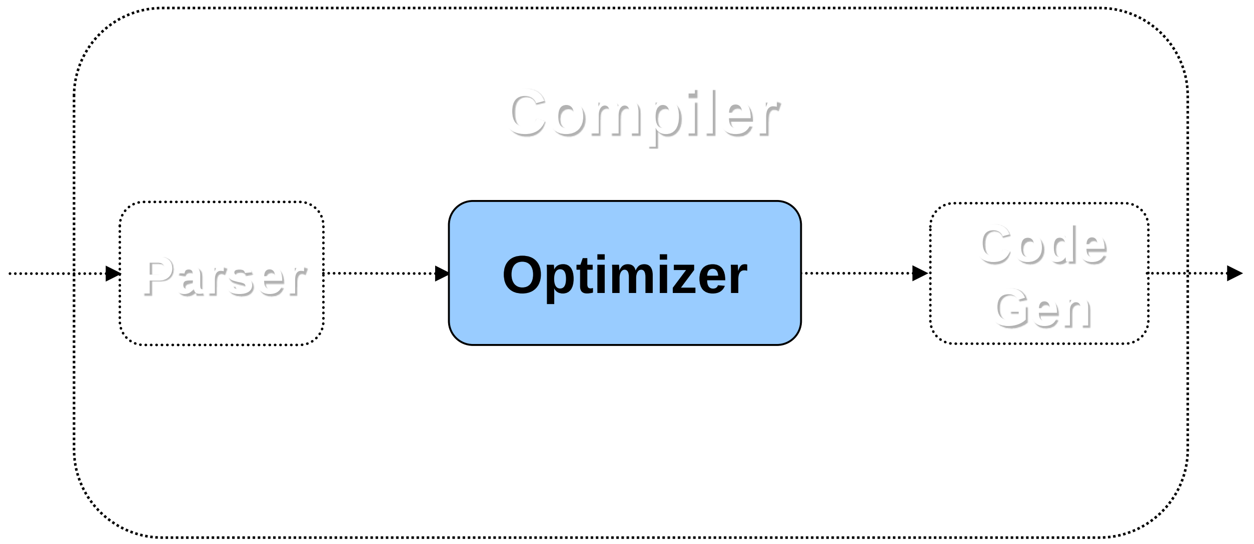


# Advanced Optimizer Design

CSE 231

Instructor: Sorin Lerner

## 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… optimizers

## What do these tools have in common?

- Bug finders
- Program verifiers
- Code refactoring tools
- Garbage collectors
- Runtime monitoring system
- And… optimizers

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
  - cornerstone of a variety of program analysis tools
  - useful no matter what your future path

- Get a feel for compiler research/implementation
  - useful if you don’t have a research area picked
  - also useful if you have a research area picked

## Course topics

- Representing programs
- 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
  - Call Graph

## Course topics (more details)

- Analysis/Transformation Algorithms
  - Dataflow Analysis
  - Interprocedural analysis
  - Pointer analysis
  - Rule-based analyses and transformations
  - Constraint-based analysis

## Course topics (more details)

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

## Course pre-requisites

- No compilers background necessary
- No familiarity with lattices
  - I will review what is necessary in class
- Familiarity with functional/OO programming
  - Optimization techniques for these kinds of languages
- Standard ugrad cs curriculum likely enough
  - Talk to me if you're concerned

## Course work

- In-class midterm (30%)
  - Date posted on web site
- Take-home final (30%)
  - Date posted on web site
- Course project (35%)
- Class readings (5%)

## Course project

- Goal of the project
  - Get some hands on experience with compilers

    **and/or** Get a feel for what research is like in PL
- Three kinds of projects:
  - **research-y**: explore some interesting ideas and try them out
  - **implementation-y**: pick some existing idea out there, and implement it
  - **paper-y**: read 10 good papers on a topic, and write a report summarizing and integrating

## Course project

- Groups of 3 (make groups by this Friday)
- Pick something to advance your personal enrichment goals
- Eg: something related to your research, something that will help your career
- Milestones
  - Project proposal (due end of week 2)
  - Mid-point status report (5 weeks in)
  - Final presentation/written report (end of quarter)

## Readings

- Paper readings throughout the quarter
- Seminal papers and state of the art
- Gives you historical perspective
- Shows you lineage from idea to practice

## Administrative info

- Class web page is up
  - http://cseweb.ucsd.edu/classes/fa15/cse231-a/
  - (or Google “Sorin Lerner”, follow “Teaching Now”)
  - Will post lectures, readings, project info, etc.

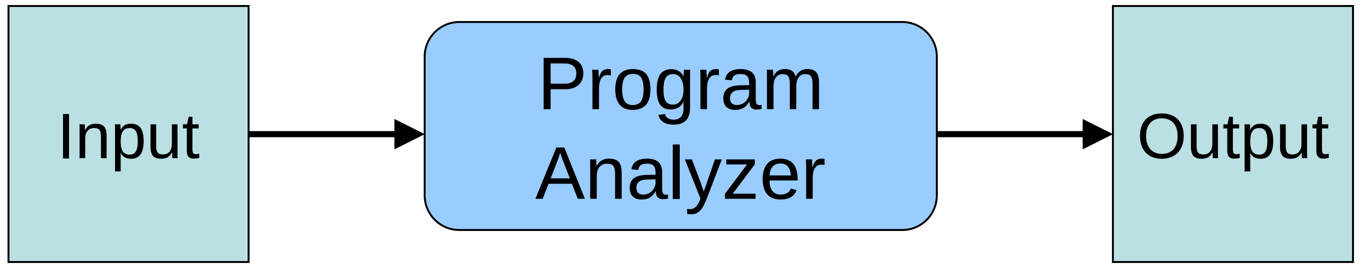
- Piazza link on web page
  - Use for questions, answers
  - Especially LLVM/project Q&A

## Academic Integrity

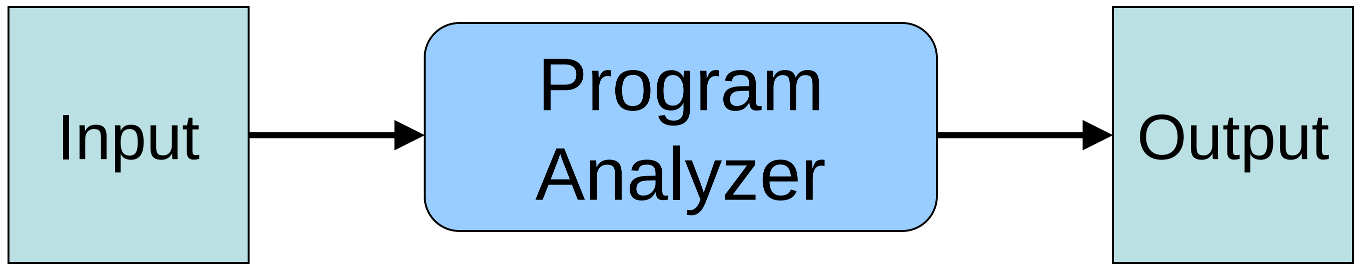
- Governed by Policy on Integrity of Scholarship (http://senate.ucsd.edu/Operating-Procedures/Senate-Manual/Appendices/2)
- Allegations are handled by Academic Integrity Office (https://students.ucsd.edu/academics/academic-integrity)
- Course penalty for any cheating in 231 will be a failing grade for the entire class
- Cheaters may be subject to additional administrative sanctions

## Questions?

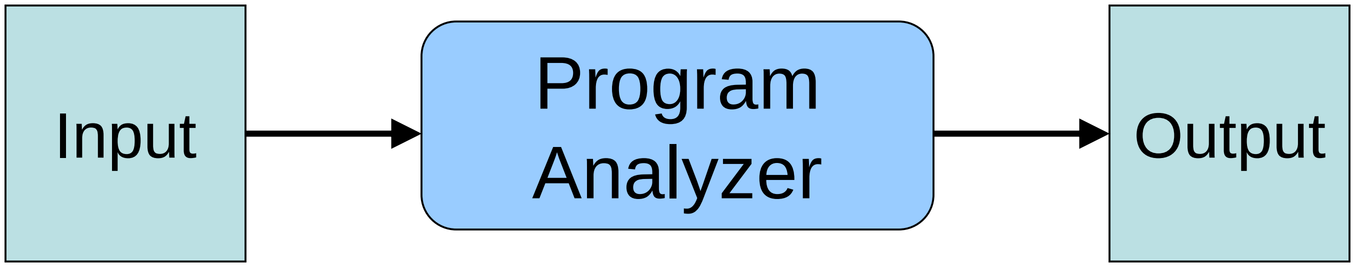
## Program Analyzer Issues (discuss)

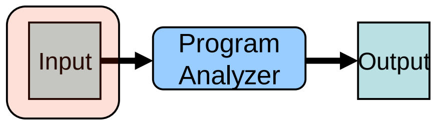


## Program Analyzer Issues (discuss)



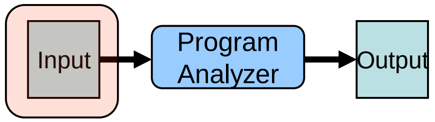
## Program Analyzer Issues (discuss)





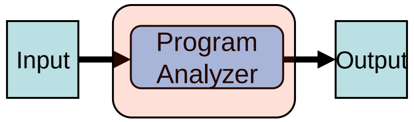
# Input issues

- 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?



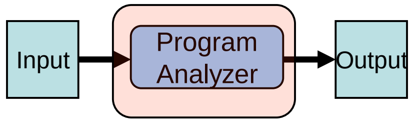
# Input issues

- 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?



# Analysis issues

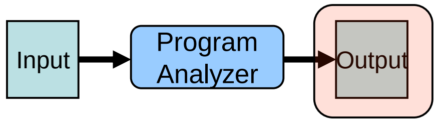
- Analysis/compilation model
  - Separate compilation/analysis
    - quick, but no opportunities for interprocedural analysis
  - Link-time
    - allows interprocedural and whole program analysis
    - but what about shared precompiled libraries?
    - and what about compile-time?
  - Run-time
    - 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



# Analysis issues

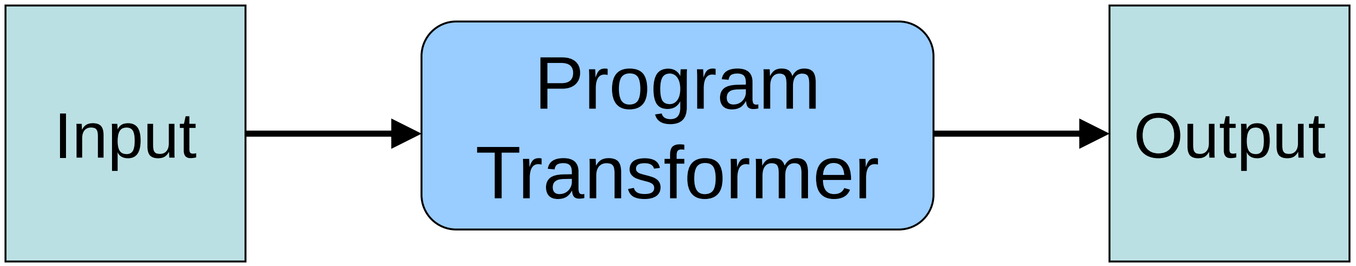
- Does running-time matter?
  - for use in IDE?
  - or in overnight compile?

## 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 algorithms for solving undecidable problems?
  - notion of approximation
  - statistical output

## Program Transformation Issues (discuss)

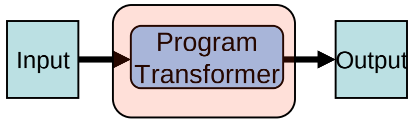


Input
Program Transformer
Output

## Input issues

- A program, and …
- Program analysis results
- Profile info?
- Environment: # of CPUs, # of cores/CPU, cache size, etc.
- 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?

Input
Program Transformer
Output

## Output issues

- Output in same IL as input?

- Should the output program behave the same way as the input program?