

Introduction to Programming Languages and Compilers

CS164
12:30-2:00 TT
60 Evans

Prof. Necula CS 164 Lecture 1

1

Aministrivia

- Course home page:
<http://www-inst.eecs.berkeley.edu/~cs164>
- If you are on the **waiting list**, follow the normal procedures (see class web page)
 - The course staff is not involved!
 - If you are **enrolled**, you don't need to do anything
- No discussion sections this week
- Pick up class accounts
 - At the end of lecture today, and from Tachio afterwards

Prof. Necula CS 164 Lecture 1

2

Course Structure

- Course has theoretical and practical aspects
- Need both in programming languages!
- Written assignments = theory
 - Class hand-in, right before lecture
- Programming assignments = practice
 - Electronic hand-in
- Strict deadlines

Prof. Necula CS 164 Lecture 1

3

Academic Honesty

- Don't use work from uncited sources
 - Including old code
- We use plagiarism detection software
 - 6 cases in last few semesters



Prof. Necula CS 164 Lecture 1

4

The Course Project

- A big project
- ... in 5 easy parts
- Start early!

Prof. Necula CS 164 Lecture 1

5

How are Languages Implemented?

- Two major strategies:
 - Interpreters (older, less studied)
 - Compilers (newer, more extensively studied)
- Interpreters run programs "as is"
 - Little or no preprocessing
- Compilers do extensive preprocessing
 - Most implementations use compilers

Prof. Necula CS 164 Lecture 1

6

(Short) History of High-Level Languages

- 1953 IBM develops the 701
- All programming done in assembly
- Problem: Software costs exceeded hardware costs!
- John Backus: "Speedcoding"
 - An interpreter
 - Ran 10-20 times slower than hand-written assembly

Prof. Necula CS 164 Lecture 1

7

FORTRAN I

- 1954 IBM develops the 704
- John Backus
 - Idea: translate high-level code to assembly
 - Many thought this impossible
- 1954-7 FORTRAN I project
- By 1958, >50% of all software is in FORTRAN
- Cut development time dramatically
 - (2 wks → 2 hrs)

Prof. Necula CS 164 Lecture 1

8

FORTRAN I

- The first compiler
 - Produced code almost as good as hand-written
 - Huge impact on computer science
- Led to an enormous body of theoretical work
- Modern compilers preserve the outlines of FORTRAN I

Prof. Necula CS 164 Lecture 1

9

The Structure of a Compiler

1. Lexical Analysis
2. Parsing
3. Semantic Analysis
4. Optimization
5. Code Generation

The first 3, at least, can be understood by analogy to how humans comprehend English.

Prof. Necula CS 164 Lecture 1

10

Lexical Analysis

- First step: recognize words.
 - Smallest unit above letters

This is a sentence.

- Note the
 - Capital "T" (start of sentence symbol)
 - Blank " " (word separator)
 - Period "." (end of sentence symbol)

Prof. Necula CS 164 Lecture 1

11

More Lexical Analysis

- Lexical analysis is not trivial. Consider:
ist his ase nte nce
- Plus, programming languages are typically more cryptic than English:
*p->f += -.12345e-5

Prof. Necula CS 164 Lecture 1

12

And More 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, ;`

Prof. Necula CS 164 Lecture 1

13

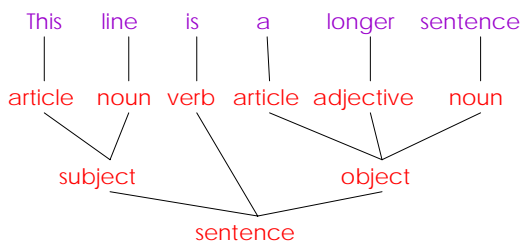
Parsing

- Once words are understood, the next step is to understand sentence structure
- Parsing = Diagramming Sentences
 - The diagram is a tree

Prof. Necula CS 164 Lecture 1

14

Diagramming a Sentence

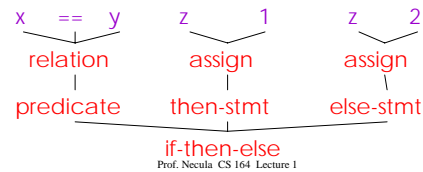


Prof. Necula CS 164 Lecture 1

15

Parsing Programs

- Parsing program expressions is the same
- Consider:
`if x == y then z = 1; else z = 2;`
- Diagrammed:



Prof. Necula CS 164 Lecture 1

16

Semantic Analysis

- Once sentence structure is understood, we can try to understand "meaning"
 - But meaning is too hard for compilers
- Compilers perform limited analysis to catch inconsistencies
- Some do more analysis to improve the performance of the program

Prof. Necula CS 164 Lecture 1

17

Semantic Analysis in English

- Example:
`Jack said Jerry left his assignment at home.`
 What does "his" refer to? Jack or Jerry?
- Even worse:
`Jack said Jack left his assignment at home?`
 How many Jacks are there?
 Which one left the assignment?

Prof. Necula CS 164 Lecture 1

18

Semantic Analysis in Programming

- Programming languages define strict rules to avoid such ambiguities
- This C++ code prints "4"; the inner definition is used

```
{  
    int Jack = 3;  
    {  
        int Jack = 4;  
        cout << Jack;  
    }  
}
```

Prof. Necula CS 164 Lecture 1

19

More Semantic Analysis

- Compilers perform many semantic checks besides variable bindings
- Example:
 Jack left her homework at home.
- A "type mismatch" between her and Jack; we know they are different people
 - Presumably Jack is male

Prof. Necula CS 164 Lecture 1

20

Optimization

- No strong counterpart in English, but akin to editing
- Automatically modify programs so that they
 - Run faster
 - Use less memory
 - In general, conserve some resource
- The project has no optimization component

Prof. Necula CS 164 Lecture 1

21

Optimization Example

$X = Y * 0$ is the same as $X = 0$

NO!

Prof. Necula CS 164 Lecture 1

22

Code Generation

- Produces assembly code (usually)
 - which is then assembled into executables by an assembler
- A translation into another language
 - Analogous to human translation

Prof. Necula CS 164 Lecture 1

23

Issues

- Compiling is almost this simple, but there are many pitfalls.
- Example: How are erroneous programs handled?
- Language design has big impact on compiler
 - Determines what is easy and hard to compile
 - Course theme: many trade-offs in language design

Prof. Necula CS 164 Lecture 1

24

Compilers Today

- The overall structure of almost every compiler adheres to our outline
- The proportions have changed since FORTRAN
 - Early: lexing, parsing most complex, expensive
 - Today: optimization dominates all other phases, lexing and parsing are cheap

Prof. Necula CS 164 Lecture 1

25

Trends in Compilation

- Optimization for speed is less interesting. But:
 - scientific programs
 - advanced processors (Digital Signal Processors, advanced speculative architectures)
 - Small devices where speed = longer battery life
- Ideas from compilation used for improving code reliability:
 - memory safety
 - detecting concurrency errors (data races)

- ...

Prof. Necula CS 164 Lecture 1

26

Trends

- Compilers
 - More needed and more complex
 - Driven by increasing gap between
 - new languages
 - new architectures
 - Venerable and healthy area

Prof. Necula CS 164 Lecture 1

27

Why Study Languages and Compilers ?

- Increase capacity of expression
- Improve understanding of program behavior
- Increase ability to learn new languages
- Learn to build a large and reliable system
- See many basic CS concepts at work

Prof. Necula CS 164 Lecture 1

28