Welcome!

COMP 520 - Compilers

Spring 2016

Jan Prins

- Course web page
  - http://www.cs.unc.edu/~prins/Classes/520
  - Please check syllabus and Piazza signup links
  - Lecture notes for today are online

- Please pick up at back of the room
  - Short assignment (single sheet, 2-sided)
    - Also available on course web page

- Reading assignment for Thu Jan 14
  - PLPJ Chapter 1
What is this course about?

- How can a program written in a modern computer programming language be run on a computer?
  - Example: execution of a C program (linux)

```
prog.c
Source program

C Compiler

cc
machine instructions

prog.o

ld
Machine code
Library

Machine code

prog.exe

executable program

input

Computer

output
```
A more detailed view of the C compiler

- Recognize legal source programs
- Issue appropriate errors for invalid programs
- Generate correct (and efficient) machine code for valid programs

... x = x + 5; ...

... 010110110 ...

source program

Compiler

machine code

errors
How does a compiler work?

- A compiler translates between computer languages
  - convert a program in the source language (e.g. C) to a program in the target language (e.g. machine instructions)
  - hopefully preserving meaning!

- How? “Syntax-directed translation”
  - by analogy to natural language translation
    - meaning is conveyed using the structure of sentences
      - subject, verb, object

- Translation steps
  - decode (discover) the “structure” from the input character stream
  - match source language concepts to target language concepts
  - encode target structure into output character stream
Example: the structure of an English sentence

- **Letters**: the dog chases a cat
- **Words**: the dog chases a cat
- **Parts of speech**: Subject: Article, Noun; Predicate: Verb; Object: Article, Noun
- **Grammar**: Subject, Predicate, Object, Sentence
Generating a translation

- **Letters:** de hond jaagt een kat
- **Words:** de hond jaagt een kat
- **Parts of Speech:** Article Noun Verb Article Noun
- **Grammar:** Subject Predicate Object
Translation is not the whole story

- We want to run the translated program!
  - execution of a Java program

```plaintext
prog.java
Source program

javac
Java compiler

prog.class
JVM instructions

Java.lang
Other class files

java prog
JVM interpreter

Computer
input

Computer
output
```
Compilers and Interpreters

• **Compiler**
  - Mechanically translates a program from one representation to another

• **Interpreter**
  - Mechanically carries out the computation specified by a program

• Program *execution* always involves a compilation step followed by interpretation
Different execution strategies

- One course objective is to understand the trade-offs involved in different execution strategies
Why study compilers and interpreters? (1)

- Understand high-level programming languages
  - what features can be translated
    - modular structure
      - classes, objects, inheritance
      - information hiding
    - user-defined (abstract) data types
    - recursive procedures

  - what features can (should) be avoided
    - features that interfere with correctness or efficiency of programs
      - incomplete type checking (unchecked casts)
      - goto statements

  - what features are not needed
    - forward declarations (header files)
    - nested procedures

- Understand compiler error and warning messages
**Example: Java generics**

**without type parameter**

```java
LinkedList list = new LinkedList();
list.add("abc");   // ok
list.add(new Foo());   // ok
String s = list.get(0);   // no!
```

**with type parameter**

```java
LinkedList<String> list = new LinkedList<String>();
list.add("abc");   // ok
list.add(new Foo());   // no!
String s = list.get(0);   // ok
```

**but ...**

here compiler issues an “unchecked cast” warning?

```java
void m(LinkedList arg) {
    LinkedList<String> t = (LinkedList<String>) arg;
    String w = t.get(0);
}
```
Why study compilers and interpreters? (2)

- Understand related tools and issues
  - Integrated Development Environments (IDEs)
    - Syntax highlighting, auto-completion
  - Debuggers
    - capabilities and limitations
  - Linkers and Loaders
    - arcane but critical in large system integration
  - Just-in-time compilers (JIT)
    - basis of efficient execution of Java and .NET
- Performance
  - Large fraction of modern performance due to advanced compilers and target machine architecture
  - But also: compiler limitations responsible for a lot of missing performance
Why study compilers and interpreters? (3)

• Useful skill
  - Many systems must parse and execute user input
    • Data base queries
    • Command lines and GUIs
  - Flexible tools are “programmable”
    • Example: grep (regular expression search)
    • Internally grep translates the regex and interprets result
  - Performance depends on sophisticated optimizing compilers
    • To get good performance, you must understand the capabilities and limitations of optimization
    • Optimization is rife with intractable and uncomputable problems
      - “Full-employment theorem” for optimizing compiler builders!
Why study compilers and interpreters? (4)

- **Pedagogical reasons**
  - Many CS concepts come together in compilers
    - **Automata theory**
      - grammars and parsing of programming language
    - **Programming language design and implementation**
      - type system and type checking, language semantics, run-time organization
    - **Data structures and algorithms**
      - used throughout the compiler
    - **Machine organization**
      - target language is (abstract) machine instructions
      - efficiency issues: caches, instruction sequences, branch prediction, ...
    - **Software engineering**
      - Compilers are large and sophisticated programs
        » can be constructed using modern design principles and patterns
      - The compiler you build in this class will likely be one of the most intricate program you have constructed

- **It’s so “meta”**
  - programs processing programs
Is this the right course for you?

- **What will we study and what is required?**
  - Let’s check the administrative handout on the course web page

- **Project**
  - Implement a compiler for a (small) subset of Java
    - generate code for a virtual machine
  - The compiler you construct will itself be a Java program
    - significant amount of Java programming, but
    - you will follow a design outlined in the text and illustrated in a sample compiler available to you
    - you will be given interfaces and specs for key parts
  - you can work in teams of two, if desired
    - a team effort earns 80% credit for each of the two members
  - there will be optional project extensions to earn additional credit!

Warning: 50% of the grade and requires a lot of programming
Evolution of Compilers: a bit of history

• The problem
  - 1954: IBM develops 704 computer (follow-on to 701)
    • All programming done in machine code (assembly) ...
    • Observation: Software development exceeded cost of hardware!

• Attempt at Solution
  - “Speedcoding”
    • An interpreter of algebraic expressions
      - Speedcode programs ran 10-20 times slower than hand-written assembly

• John Backus’ idea
  - A program to translate high-level algebraic expressions into machine instructions
    • Many thought it impossible
  - 1954-57: FORTRAN I project
    • By 1958, > 50% of all projects used FORTRAN for programming
    • Cut development time in half
• The first “compiler”
  - Etymology of the term “compiler”
    • compile:
      - to put together or compose from materials gathered from several sources
    • compiler
      - originally a program that put together different machine-language subroutines
        » a linking-loader
    • “algebraic compiler” original name of Backus’ system in 1954
      - provided rudimentary translation of algebraic expressions
      - algebraic translation aspect dropped from name over time
  
• Huge impact on programming languages and computer science
  - Led to enormous body of theoretical work on compilation
    • parsing, static analysis of programs
  - Enabled thousands of high-level languages to be proposed
    • few survive today ... (but Fortran is among them)
Things To Do

• Check course web page
  - follow Piazza link to sign up

• Start reading assignment
  - Skim 24 pages

• Short problem set
  - Just 3 simple problems, but due next Thursday

• Get set up to use CS facilities
  - You need a departmental login
    • This should have been set up for everyone already

• Remember to check the course web page

• See me if you have questions