Welcome!

COMP 520 - Compilers

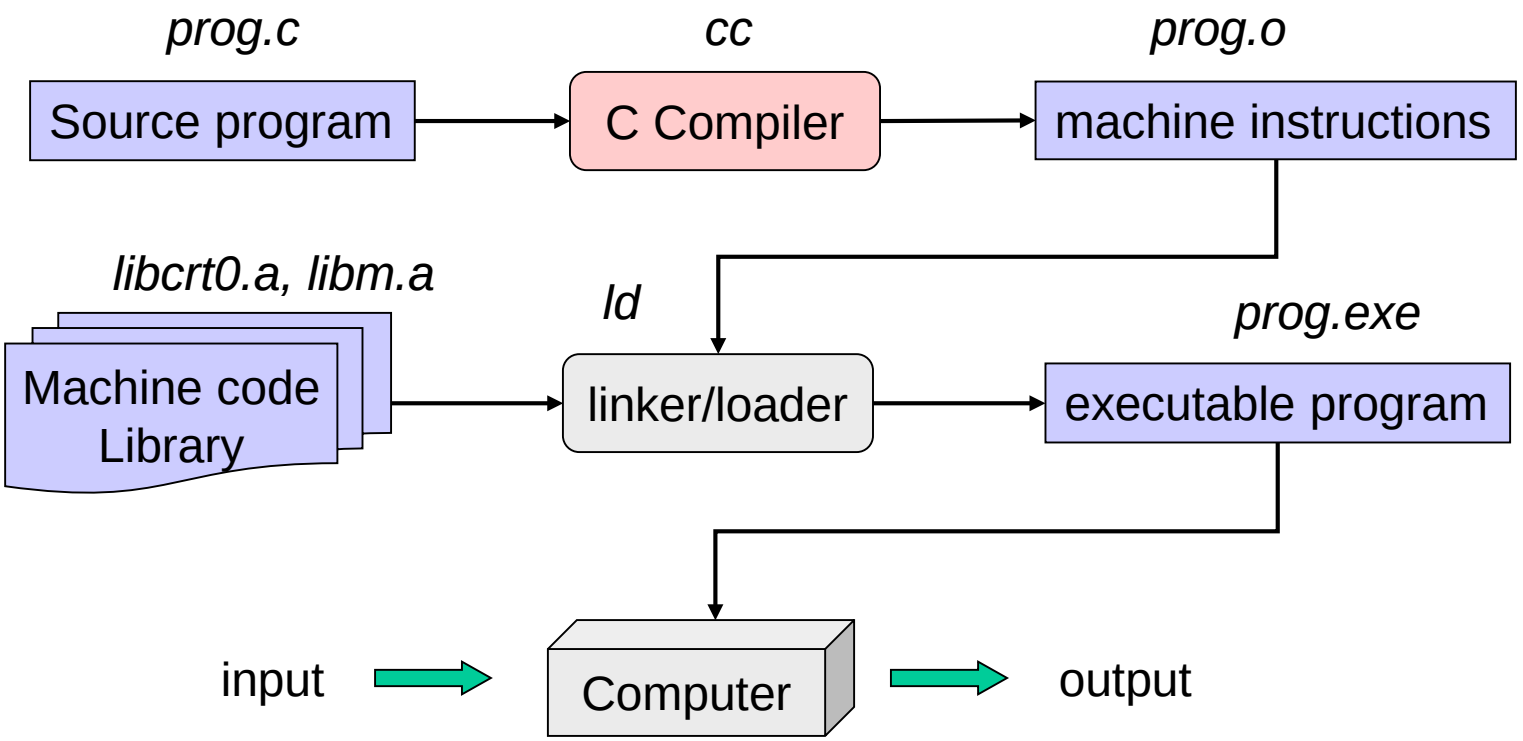
TR: 2:00 – 3:15 Spring 2022

Instr: Jan Prins, TA: Tao Tao

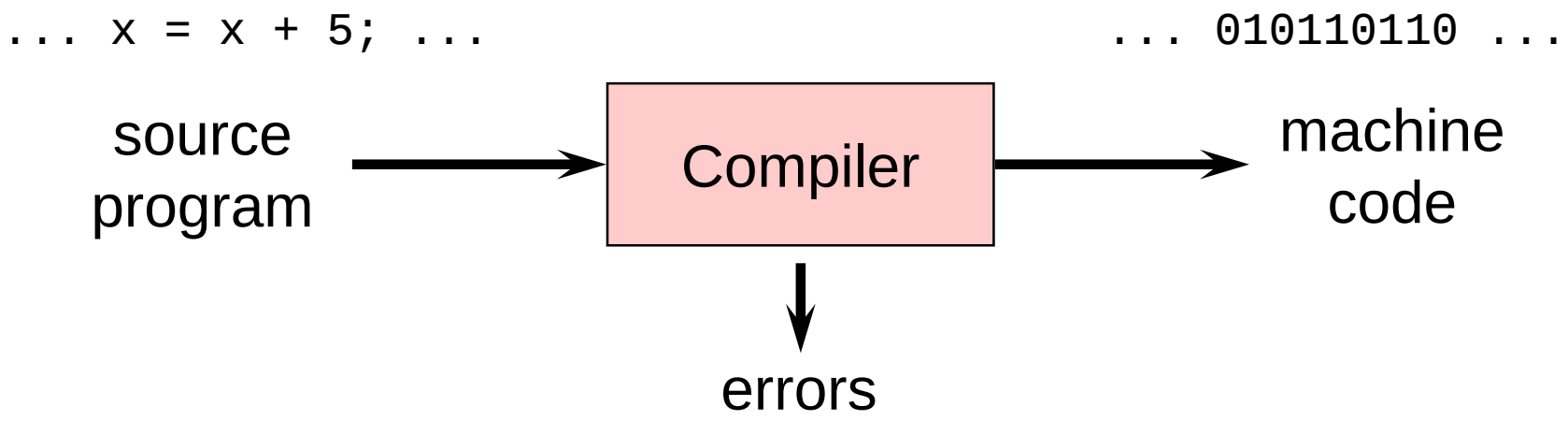
- Course web page
  - http://www.cs.unc.edu/~prins/Classes/520
  - Please check syllabus
  - Lecture slides for today are online
- Written assignment
  - Short assignment WA1 due at start of next class
    - available on course web page
- Reading assignment for Thu Jan 13
  - PLPJ Chapter 1

What is this course about?

- How do programs written in a modern computer programming language get compiled and run on a computer?
  - Example: execution of a C program (linux)



A more detailed view of the C compiler

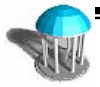


- Recognize legal source programs
- Issue appropriate errors for invalid programs
- Generate correct (and efficient) machine code for valid programs

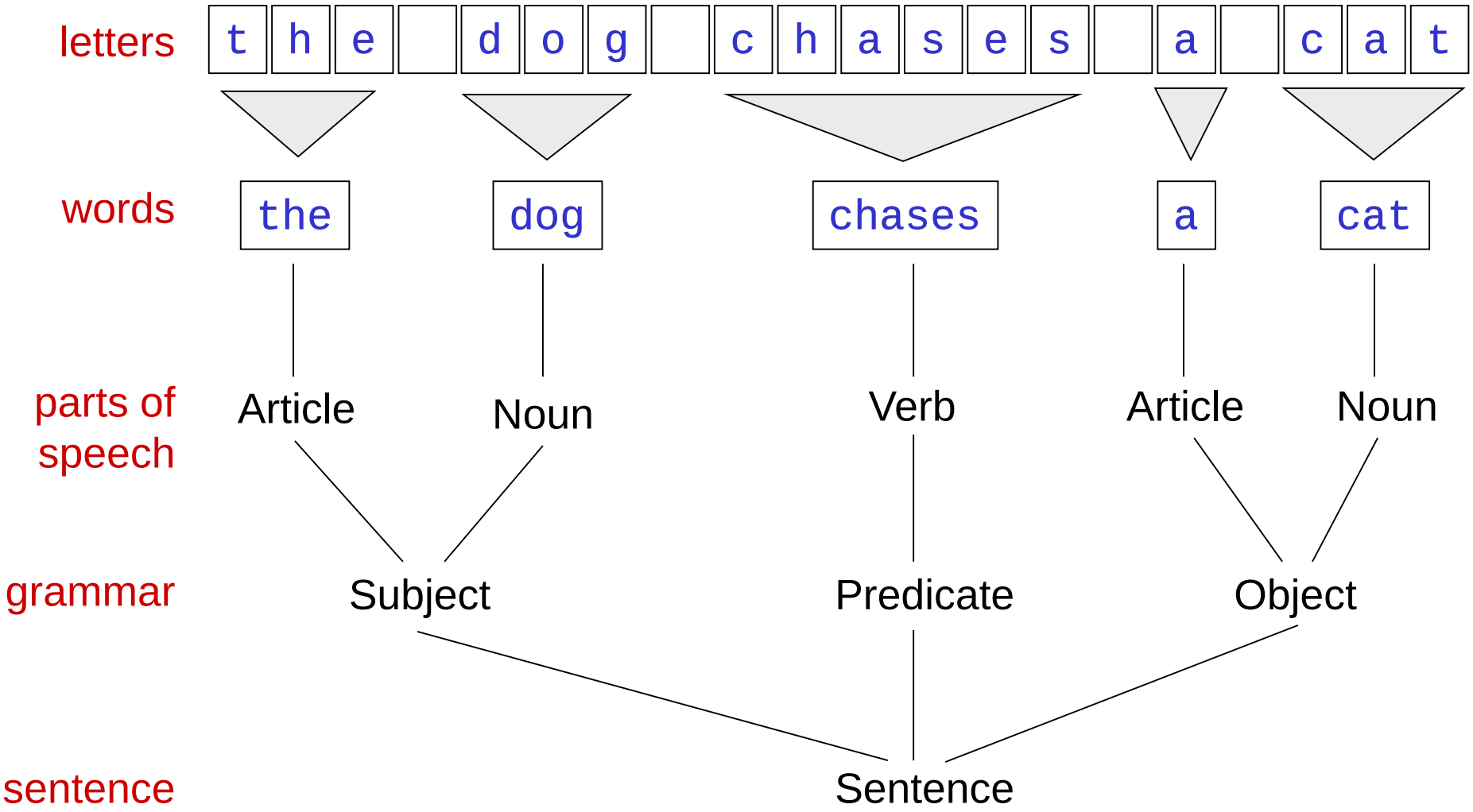
How does a compiler work?

- 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!

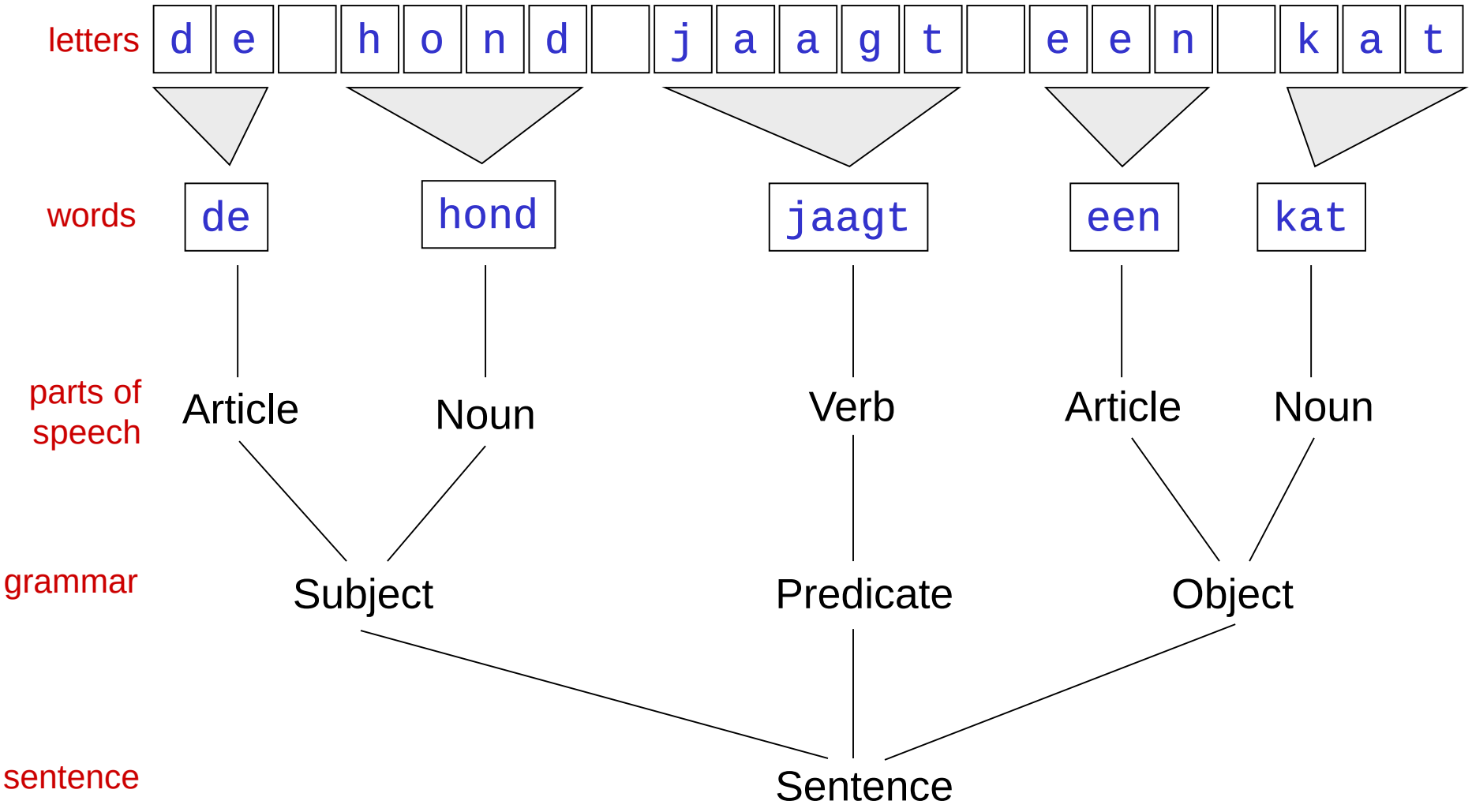
- How? "Syntax-directed translation"
  - by analogy to natural language translation
    - meaning is conveyed using the structure of sentences
      - subject, verb, object
  - Translation steps
    - decode (discover) the "structure" from the input character stream
    - match source language concepts to target language concepts
    - encode target structure into output character stream



Example: the structure of an English sentence

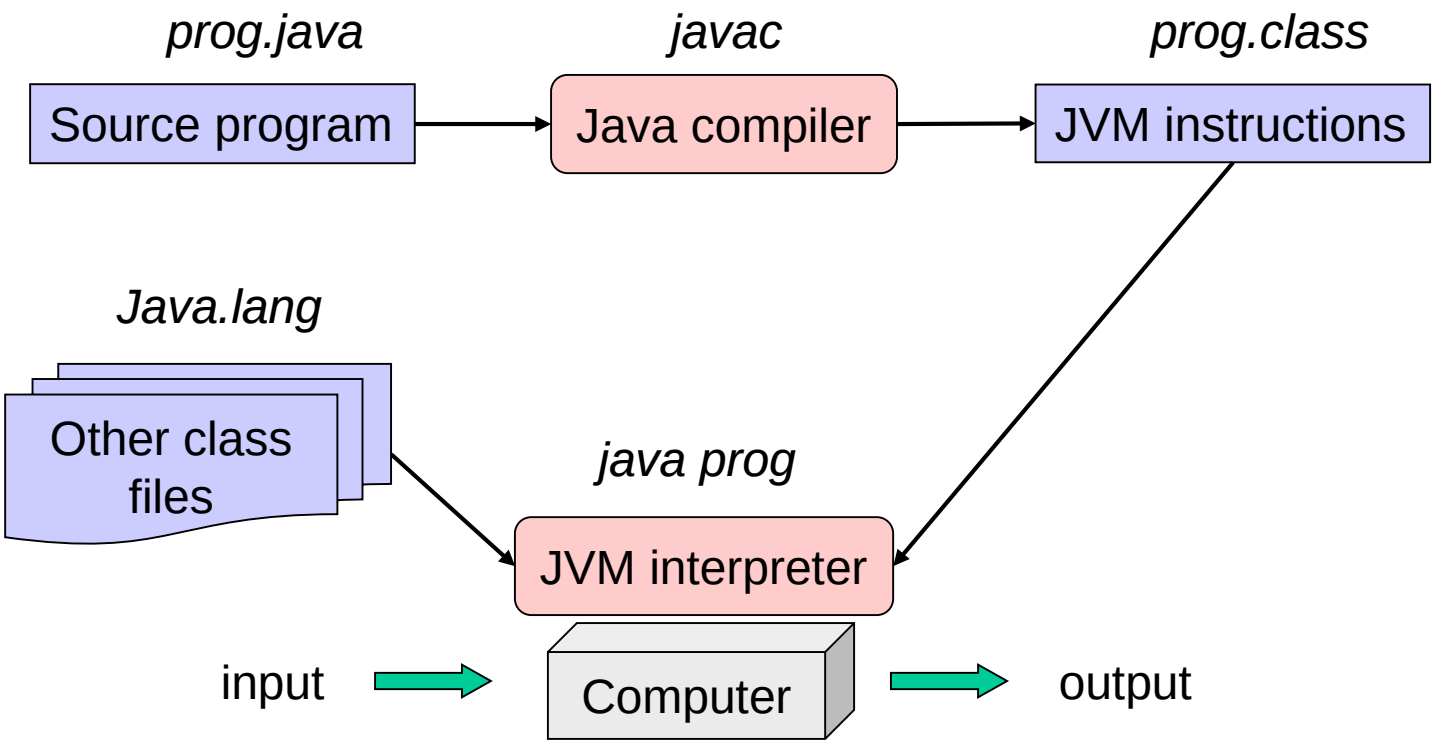


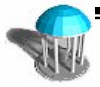
Generating a translation



Translation is not the whole story

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



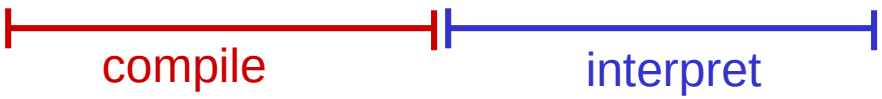


Compilers and Interpreters

- 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

C:

C compiler — Intel processor

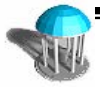
Java:

Java compiler — Java virtual machine

JavaScript: (originally)

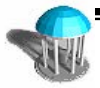
parser — tree evaluation

- 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

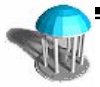
without type parameter

```
LinkedList list =
   new LinkedList();

list.add("abc");       // ok
list.add(new Foo()));  // ok

String s = list.get(0); // no!
```

with type parameter

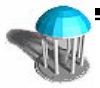
```
LinkedList<String> list =
   new LinkedList<String>();

list.add("abc");       // ok
list.add(new Foo()));  // no!

String s = list.get(0); // ok

void m(LinkedList arg) {
  LinkedList<String> t =
   (linkedList<String>) arg;
  String w = t.get(0);
}
```

but …
here compiler issues an "unchecked cast" warning?



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, runtime systems, and target machine architectures
    - But also: compiler limitations responsible for a lot of missing performance



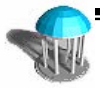
Why study compilers and interpreters? (3)

- 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 reg expr 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 recognizing automata of formal languages
    - Programming language design and implementation
      - type system and type checking, language semantics, run-time organization
    - Data structures and algorithms
      - used within a compiler
    - Machine organization
      - target language is a (virtual or real) machine
      - efficiency issues: caches, register allocation, instruction sequences …
    - Software engineering
      - Compilers are large and sophisticated programs
        - » can be constructed using modern design principles and patterns
      - The compiler you build in this class may well be the most intricate program you have constructed!

- It's so "meta"
  - programs processing programs

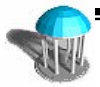


Is this the right course for you?

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

~50% of your grade and a lot of programming!

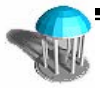
- Project
  - implement a compiler for a (small) subset of Java
    - generate code for a virtual machine
  - the compiler you construct will itself be a Java program
    - significant amount of Java programming, but
    - you will follow a design outlined in the text and illustrated in a sample compiler 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!



A message from the Registrar …

- Starting with the Spring Semester 2020, all students will be required to confirm that they have reviewed the Honor Code and that they have begun academic activity for all registered courses at Carolina.
- The acknowledgement in activity helps UNC fulfill a federal requirement for participation in Title IV (student financial aid) programs and helps us more effectively serve students.
- Students will receive an email with instructions on how to complete this task each semester in Connect Carolina.

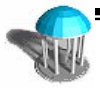
- A student's failure to participate could bring significant consequences:
  - A student's failure to acknowledge that they have begun a course may have an impact on future registration, including prevention from registering in the next term.
  - Students receiving financial aid may see their awards lowered or removed completely.



Triangle Examples (1)

- Triangle commands
  - Conditional command
  - Scope command

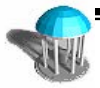
```
if x > y then
    let const xcopy ~ x
    in
        begin
            x := y;
            y := xcopy
        end
else
```

Triangle Examples (2)

- Triangle expressions
  - Scoped expression
  - Conditional expression

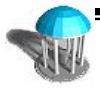
```
let
    const taxable ~ if income > allowance
                    then income – allowance
                    else 0
in
    taxable / 4
```

Triangle Examples (3)

- Triangle types, procedures, and operators
  - Named type
  - Function declaration
  - Operator declaration

```
type Point ~ record
                x: Integer, y: Integer
             end;

func projection (pt: Point) : Point ~
   {x ~ pt.x, y ~ 0 – pt.y};

func /\ (b1: Boolean, b2 : Boolean) : Boolean ~
   if b1 then b2 else false
```

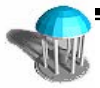



Evolution of Compilers: a bit of history

- 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

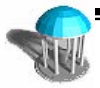
- John Backus' idea
  - A program to *translate* high-level algebraic expressions into machine instructions
    - Many thought it impossible
  - 1954-57: FORTRAN I project
    - By 1958, > 50% of all projects used FORTRAN for programming
    - Cut development time in half



FORTRAN I

- 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
  - source for all information – check regularly
  - follow Piazza link to sign up and use Piazza for questions
- Start reading assignment
  - Skim 24 pages and start looking at chapter 3
- Short problem set
  - Just 3 simple questions, write answers on handout, due Thursday
- Get set up to use course facilities
  - Details TBD
- Look ahead
  - Start looking at chapter 3, and preview of first project phase
  - Look at simpleScannerParser example (will be placed online)