

198:314 Principles of Programming Languages

Course Goals

- To gain understanding of the basic structure of programming languages:
 - Data types, control structures, naming conventions,...
- To study different language paradigms:
 - Functional (*Scheme*), imperative (*C*), logic (*Prolog*), object-oriented (*Java*), event-driven (*Java*)
 - To ensure an appropriate language is chosen for a task
- To know the principles underlying all programming languages:
 - To make learning new programming languages easier
 - To enable *full* use of a programming language

Programming languages are tools \Rightarrow understand how to design or use them

Course Information

Prerequisites:

- CS 112 (Data Structures)
- CS 205 (Introduction to Discrete Structures), or
EE 227

Important facts:

staff: Prof. Ulrich Kremer
TA Jerry Hom + grader (TBA)

lectures: MW 2:50-4:10p.m., WL-AUD

recitations: attendance mandatory,
will not start until next week

books on reserve: Science & Engineering Resource Center
(SEC), Busch Campus

ROOM CHANGES UNTIL SEPT. 23!
SEE OUR WEB PAGE

Basis for grades (subject to changes):

10% homework
25% mid-term exam (Oct. 22, closed book)
35% final exam (Dec. 19, closed book, cumulative)
30% four programming projects

Course Information (Cont.)

- The textbook for this course is “*Programming Languages, Principles and Practices*” by Kenneth C. Loudon, 2nd Edition, Thomson-Books/Cole, 2003.
- Additional (recommended) texts: see course web page

Course material is available on the World Wide Web at

<http://remus.rutgers.edu/cs314>

In addition, there is a news group for this class:

ru.nb.dcs.class.314

**YOU SHOULD READ THE NEWS GROUP
AND LOOK AT THE HOME PAGE AT
LEAST EVERY OTHER DAY**

All programming will be done on the **undergraduate UNIX Cluster** (remus and romulus). Get yourself a **Unix** account. Learn to do the normal things — edit, compile, ...

Course Information (Cont.)

IMPORTANT \Rightarrow see our 314 web page!

- Failure to take a scheduled exam
- Grading of homeworks and projects
- Submission of programming projects using "handin"
- Partial credit for late submissions
- Academic Integrity

Email your instructor or TA

- **Subject line** has to start with **314:**,
e.g., *314: Copy of my transcript*
- **No** project and homework questions; please post them on the news group;

Course Information (Cont.)

Special permission numbers

- common policy across all 9 sections (Ryder, Kremer)
- priority to graduating seniors; **NEED COPY OF YOUR TRANSCRIPT!**
- put your name on the SP request list distributed in class
- first get into the course; we will deal with switching sections later!
- will distribute SP numbers via email, most likely by the end of this week

Course Information (Cont.)

I use overhead transparencies

- I try to moderate my speed
- *You* need to say STOP!
- all transparencies are on the Web (PostScript, PDF)
- you should still take some notes

I'll tell you where we are in the book

- I don't lecture directly from the book
- *You* need to read the book
- I consider coming to the lecture in addition to the recitations mandatory

What is the Purpose of a Programming Language?

A programming language is ...

a set of conventions for communicating an algorithm. *Horowitz*

Purposes:

- specifying algorithm and data
- communicating to other people
- establishing correctness

Why Use Anything Besides Machine Code?

- Readable, familiar notations
- Machine independence (portability)
- Consistency checks during implementation
- Acceptable loss of efficiency

First FORTRAN compiler built by IBM, in 1957, translated into code as efficient as hand-coded code. *John Backus*

- Dealing with scale

The art of programming is the art of organizing complexity. *Dijkstra, 1972*

Why Learn More than One Programming Language?

- Each language encourages thinking about a problem in a particular way.
- Each language provides slightly different functionality.

⇒ The language should match the problem.

Why Learn About Programming Language PRINCIPLES?

A programming language is a **tool**.

Studying the design of a tool leads to:

- Better understanding of its functionality and limitations.
- Increased competence in using it.
- Basis for lots of other work in computer science.

Computational Paradigms

Imperative:

Sequence of state-changing actions.

- Manipulate an abstract machine with:
 1. Variables naming memory locations
 2. Arithmetic and logical operations
 3. Reference, evaluate, assign operations
 4. Explicit control flow statements
- Fits the von Neumann architecture closely
- Key operations: *Assignment* and “*Goto*”

Functional:

Composition of operations on data.

- No named memory locations
- Value binding through parameter passing
- Key operations: *Function application* and *Function abstraction*

Basis in **lambda calculus**

Computational Paradigms (Cont.)

Logic:

Formal logic specification of problem.

- Programs say *what* properties the solution must have, not *how* to find it
- Solutions through reasoning process.
- Key operation: *Unification*

Basis in first order predicate logic

Object-Oriented:

Communication between abstract objects.

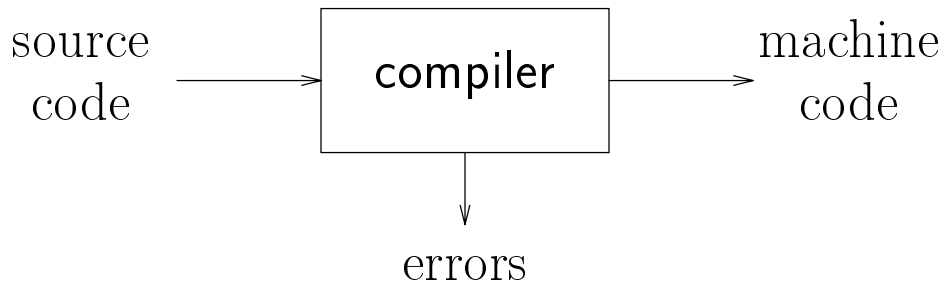
- “Objects” collect both the data and the operations
- “Objects” provide *data abstraction*
- Can be either imperative or functional
- Key operation: *Message passing or Method invocation*

Event-Driven:

Objects are associated with events

- events are asynchronous
- arrival of an event triggers action
- main applications: GUI, simulations
- Key operation: *event handling*

Compilers



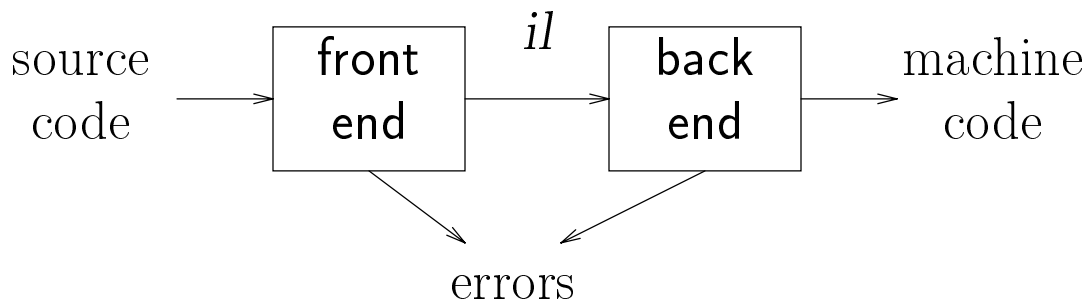
Implications:

- recognize legal (and illegal) programs
- generate correct code
- manage storage of all variables and code
- need format for object (or assembly) code

Big step up from assembler – higher level notations

Traditional two pass compiler

Pass: reading and writing entire program



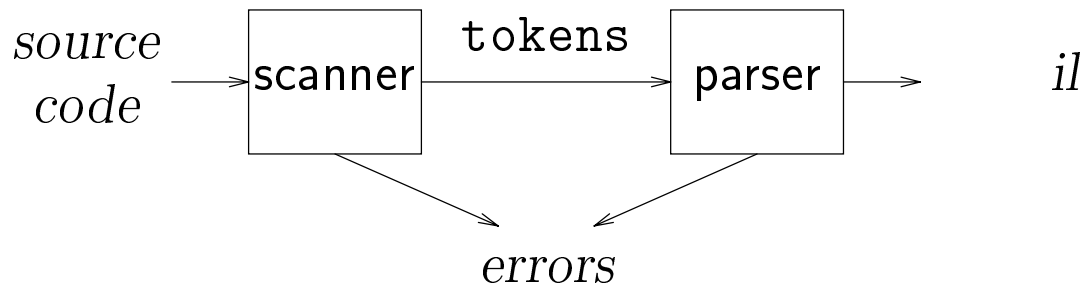
Implications:

- intermediate language (*il*)
- front end maps legal code into *il*
- back end maps *il* onto target machine
- simplify retargeting
- allows multiple front ends
- multiple passes \Rightarrow better code

Front end is $O(n)$

Back end is NP-Complete

Front end



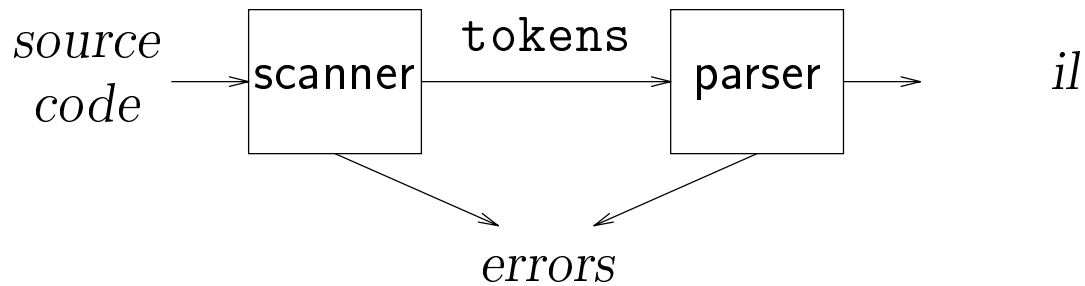
Parser: *syntax & semantic analyzer, il code generator*
(*syntax-directed translator*)

Front End Responsibilities:

- *recognize legal programs*
- *report errors*
- *produce il*
- *preliminary storage map*
- *shape the code for the back end*

Much of front end construction can be automated

Scanner



Scanner

- *maps characters into tokens – the basic unit of syntax*

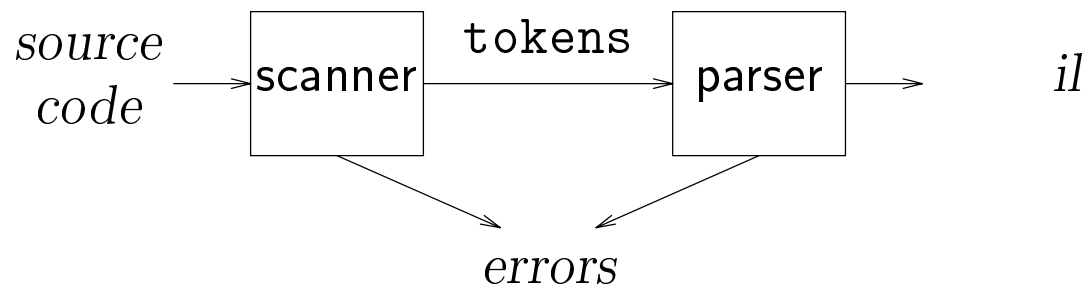
`x = x + y;`

becomes

`<id, x> = <id, x> + <id, y> ;`

- *character string for a token is a lexeme*
- *typical tokens: number, id, +, -, *, /, do, end*
- *eliminates white space (tabs, blanks, comments)*
- *a key issue is speed*
 \Rightarrow *use specialized recognizer (lex)*

Parser



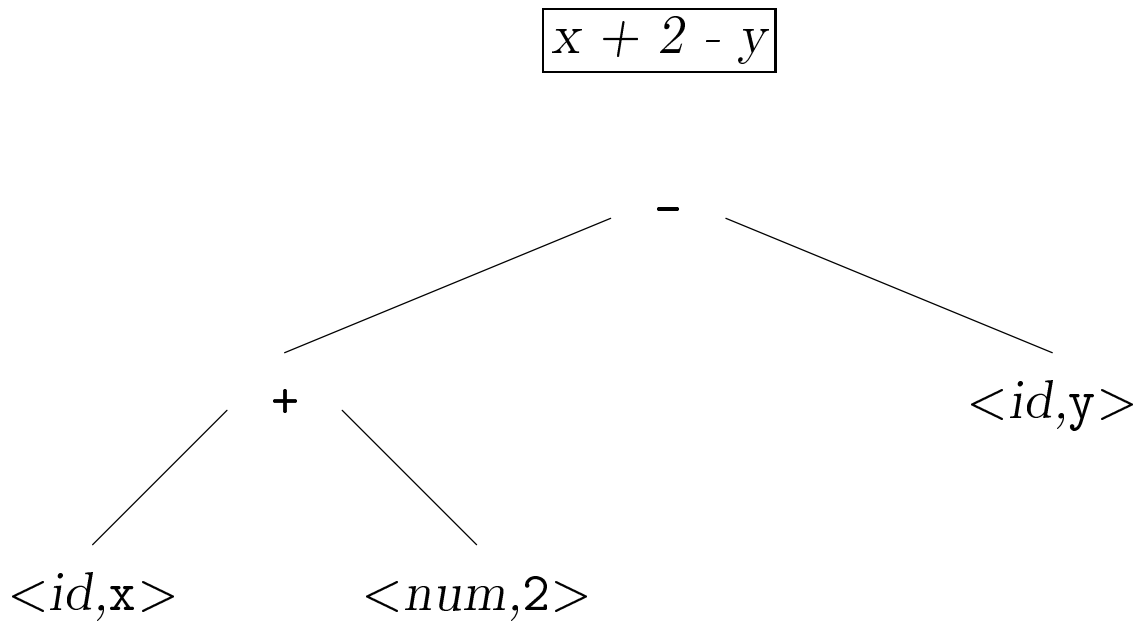
Parser:

- *recognize context-free syntax (Context Free Grammars)*
- *guide context-sensitive analysis*
- *construct $il(s)$*
- *produce meaningful error messages*
- *attempt error correction*

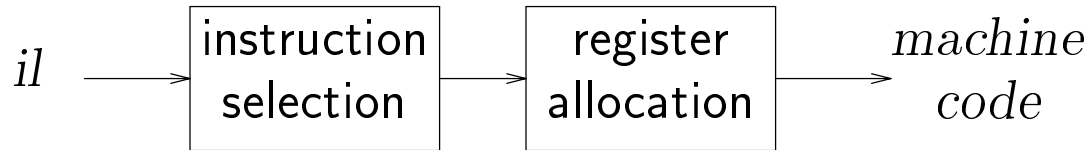
Parser generators mechanize much of the work

Example *il*: Abstract syntax tree (AST)

Compilers often use an abstract syntax tree.



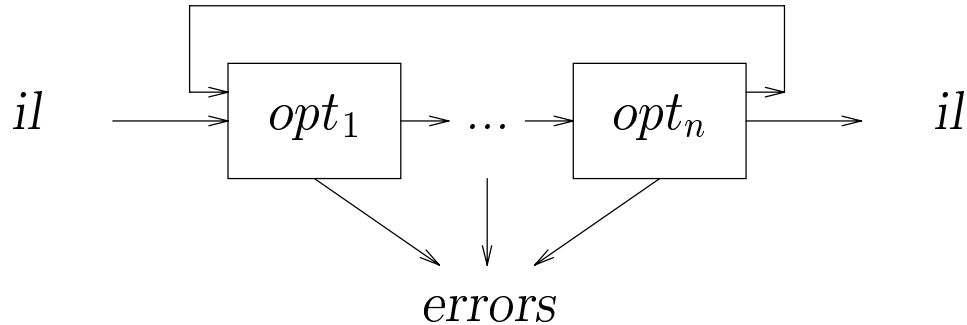
Back end



Responsibilities

- *translate il into target machine code*
- *choose instructions for each il operation*
- *decide what to keep in registers at each point*

Optimizer (middle end)



Modern optimizers are usually built as a set of passes.

Typical passes

- *discover & propagate constant values*
- *reduction of operator strength*
- *common subexpression elimination*
- *redundant computation elimination*
- *encode an idiom in some powerful instruction*
- *move computation to less frequently executed place (e.g., out of loops)*

Things to Do

Things to do for next lecture:

- *read Louden Ch 4.1; Aho-Sethi-Ullman (Dragon Book); Chs 2.2, 3.3, 3.4*
- *learn to use a Web browser such as Netscape*
- *learn to read news groups and post messages*

Recitations will start Monday next week.