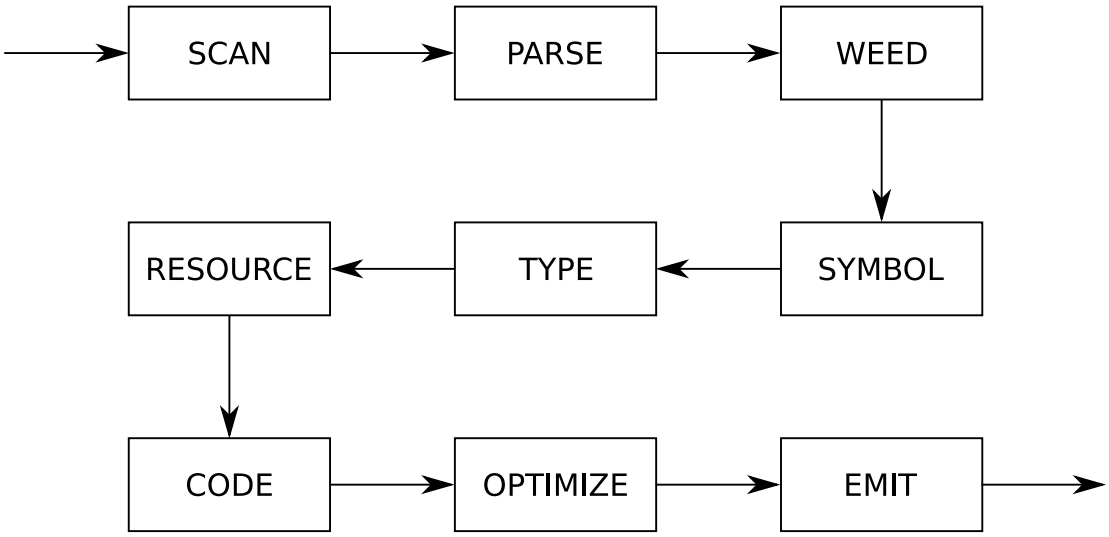
Introduction

COMP 520: Compiler Design

Eric Bodden

MWF 11:35-12:35 ENGTR 0060



Purpose:

- This course covers modern compiler techniques and their application to both general-purpose and domain-specific languages.
- The practical aspects focus on current technologies, primarily Java and interactive web services.

Contents:

- *Deterministic parsing:* Scanners, LR parsers, the `flex`/`bison` and `SableCC` tools.
- *Semantic analysis:* abstract syntax trees, symbol tables & attribute grammars, type checking, resource allocation.
- *Virtual machines and run-time environments:* stacks, heaps, objects.
- *Code generation:* resources, templates, optimizations.
- *Surveys on:* native code generation, static analysis, ...

Schedule:

- Lectures: 3 hours/week.

Prerequisites:

- COMP 273, COMP 302, (COMP 330), ability to read and write "large" programs.
- Students without COMP 330 should read the background material indicated in Week 1 of the web page ASAP.

Lecturer:

- Eric Bodden, McConnell 234,
  Office Hours MW 12:30–13:30

TA:

- Reehan Shaikh, McConnell 202,
  Office Hours TBA

Midterm: Monday, Oct. 27th (?)

Final exam: date will be announced later on

Marking Scheme:

- 10% midterm, 25% final exam, 65% assignments and project
- the 65% for assignments and projects will be divided as follows:
  - 15% for the first 3 JOOS deliverables (5% each)
  - 10% for the JOOS peephole optimizer
  - 20% for content submitted at milestones
  - 20% for the final WIG compiler and report
- Group members may be given different grades on the project work if the contributions are not reasonably equal.
- You have three late days for the term. Late days can be taken at any deadline *but not at the final one.*

Academic Integrity:

- McGill University values academic integrity. Therefore all students must understand the meaning and consequences of cheating, plagiarism and other academic offences under the Code of Student Conduct and Disciplinary Procedures.

  `http://www.mcgill.ca/integrity/studentguide/`

- In terms of this course, part of your responsibility is to ensure that you put the name of the author on all code that is submitted. By putting your name on the code you are indicating that it is completely your own work.

- If you use some third-party code you must have permission to use it and you must clearly indicate the source of the code.

Course material:

- course pack readings (readings C1-C6);
- online readings (readings O1-O5);
- slides for the lectures; and
- extensive documentation on the course web pages.

The course pack and the online readings:

- are mainly background reading;
- do not discuss the JOOS and WIG projects used in this course; and
- are required for the exercises.

The slides:

- are quite detailed; and
- are available online just before class in 1-up and 4-up formats.

The web pages:

- aim to contain all information;
- provide on-line documentation; and
- may be updated frequently.

New this year: The newsgroup

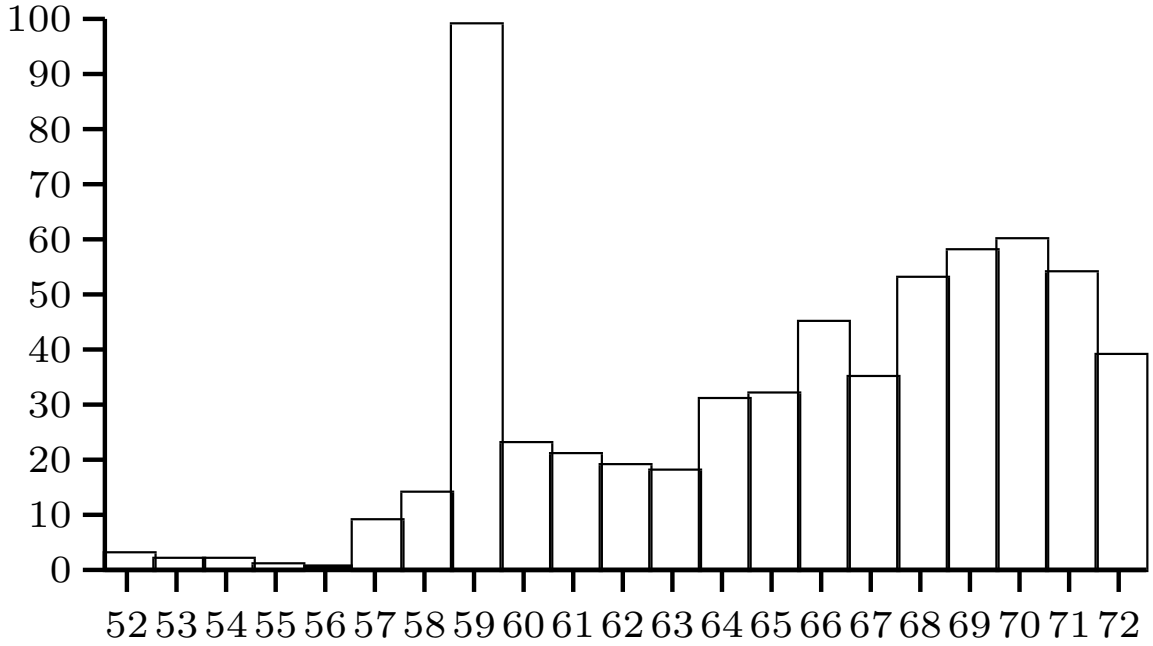
`http://groups.google.com/group/comp-520-08`

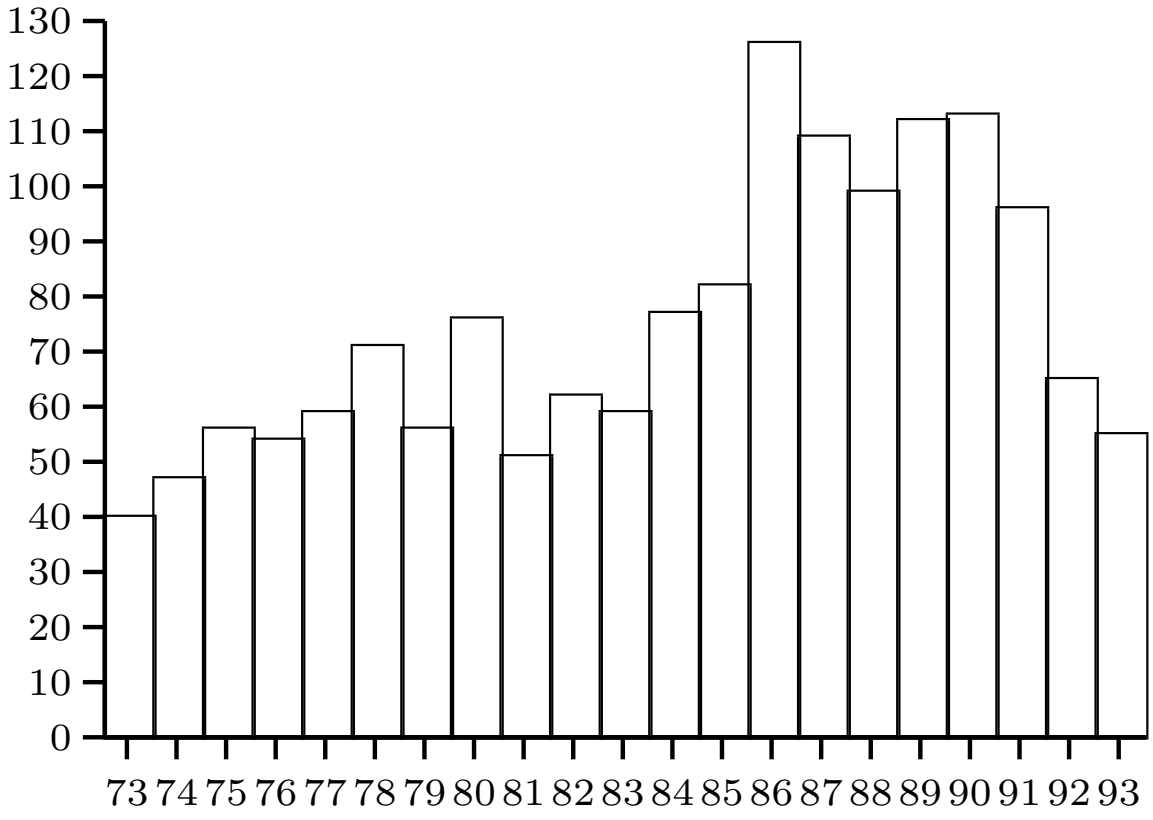
- Restricted to COMP 520 students, TA and the instructor
  - Visit group and ask for an invitation to join
  - We will then approve your invitation
- Main purpose is to discuss:
  - Problems you may encounter with the tools you need to use, and
  - Questions about the previous lecture(s)
- Do *not* use this group for:
  - Questions regarding solutions to exercises, assignments or your projects (see/email the TA instead)
  - Anything personal, etc. (see TA or instructor)

Subversion (SVN)

- Versioning system
- Good for maintaining source code but also for other things
- You will use it for your submissions
- You can also use it to track changes on the website and to access your source code templates
- Tasks for this week
  - Read SVN documentation on the website
  - Pair up with group mates (may use discussion group)
  - Send SSH key and name(s) of group mate(s) to the TA (see website)

New programming languages per year:





The compiler for the FORTRAN language:

- was implemented in 1954–1957;
- was the world's first compiler;
- was motivated by the economics of programming;
- had to overcome deep skepticism;
- paid little attention to language design;
- focused on efficiency of the generated code;
- pioneered many concepts and techniques; and
- revolutionized computer programming.

```
C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL
  LISTING
      READ INPUT TAPE 5, 501, IA, IB, IC
  501 FORMAT (3I5)
C IA, IB, AND IC MAY NOT BE NEGATIVE
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C IS GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO
      IF (IA) 777, 777, 701
  701 IF (IB) 777, 777, 702
  702 IF (IC) 777, 777, 703
  703 IF (IA+IB-IC) 777,777,704
  704 IF (IA+IC-IB) 777,777,705
  705 IF (IB+IC-IA) 777,777,799
  777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
     +     (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,
     F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```

General-purpose languages:

- allow for arbitrarily useful programs to be written
- in the theoretical sense are all Turing-complete; and
- are the focus of most programming language courses.

Prominent examples are:

- C
- C++
- FORTRAN
- Java
- ...

General purpose languages fairly obviously require full-scale compiler technology to run efficiently.

Domain-specific languages:

- extend software design; and
- are concrete artifacts that permit representation, optimization, and analysis in ways that low-level programs and libraries do not.
- They may even be visual! (e.g. boxes & arrows)

Prominent examples are:

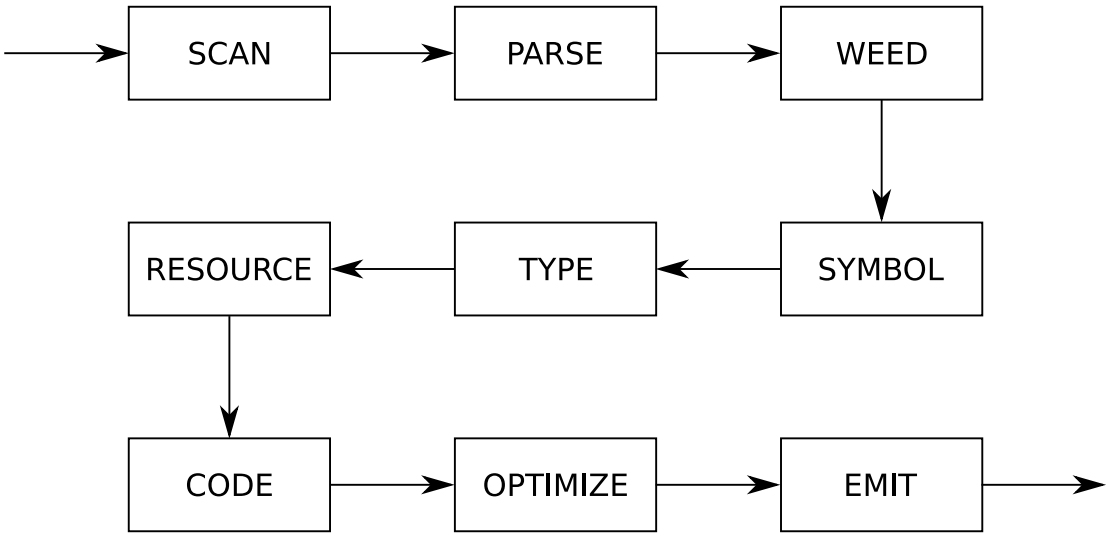
- LaTeX
- `yacc` and `lex`
- Makefiles
- HTML
- SVG
- ...

Domain-specific languages also require full-scale compiler technology.

Reasons to learn compiler technology:

- understand existing languages;
- appreciate current limitations;
- talk intelligently about language design;
- implement your very own general purpose language; and
- implement lots of useful domain-specific languages
  - encoding everything in XML is not always the best way to go
    (see Makefiles vs. Ant's build.xml)

The phases of a modern compiler:



The individual phases:

- are modular software components;
- have their own standard technology; and
- are increasingly being supported by automatic tools.

Advanced backends may contain an additional 5–10 phases.

The  project:

- Java's Object-Oriented Subset
- is compiled to Java bytecode;
- illustrates a general purpose language;
- allows client-side programming on the web;
- is used to teach by example;
- has A− source code available;
- and will be upgraded by you into an A+ version.

The  project:

- Web Interface Generator
- is compiled to C-based CGI-scripts (or other targets...);
- illustrates a domain-specific language;
- allows server-side programming on the web;
- is used to get hands-on experience;
- and will be implemented from scratch, by you!

The top 10 list of reasons why we use C for compilers:

10) it's tradition;

9) it's (truly) portable;

8) it's efficient;

7) it has many different uses;

6) ANSI C will never change;

5) you must learn C at some point;

4) it teaches discipline (the hard way);

3) methodology is language independent;

2) we have `flex` and `bison`; and

1) you can say that you have implemented a large project in C.

The top 10 list of reasons why we use Java for compilers:

10) you already know Java from previous courses;

9) run-time errors like null-pointer exceptions are easy to locate;

8) it is relatively strongly typed, so many errors are caught at compile time;

7) you can use the large Java library (hash maps, sets, lists, ...);

6) Java bytecode is portable and can be executed without recompilation;

5) you don't mind slow compilers;

4) it allows you to use object-orientation;

3) methodology is language independent;

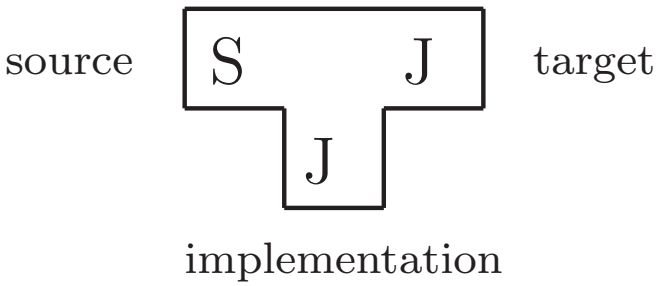
2) we have `sablecc`, developed at McGill; and

1) you can say that you have implemented a large project in Java.

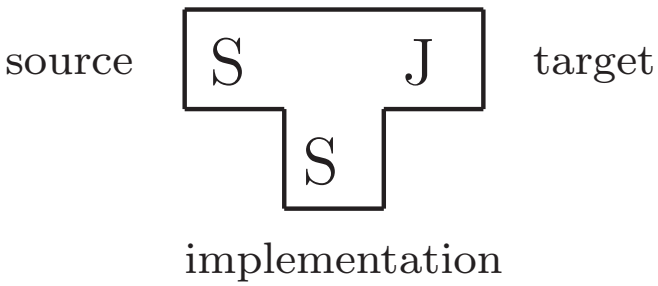
How to bootstrap a compiler (SCALA example):

- we are given a source language (L in the reading), say SCALA; and
- a target language (M in the reading), say Java.

We need the following:



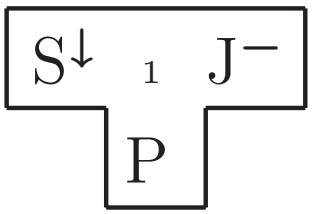
Of course, actually we like SCALA much better than Java and would therefore rather implement SCALA in itself:



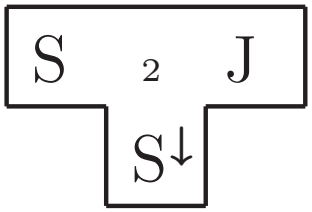
Define the following:

- $S^{\downarrow}$ is a simple subset of SCALA;
- J^{-} is inefficient Java code, and
- P is our favourite programming language, here "Pizza".

We can easily implement:

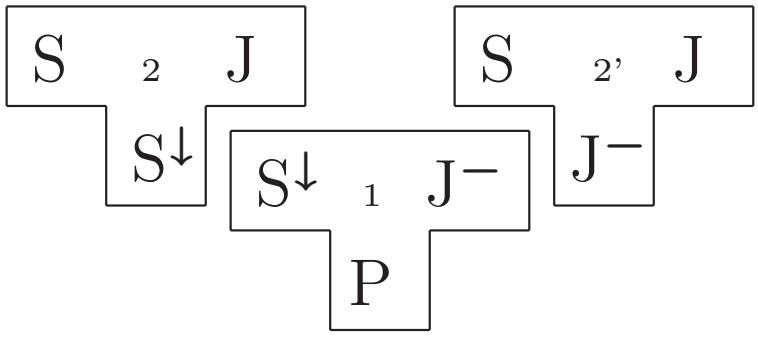


and in parallel, using $S^{\downarrow}$, we can implement:



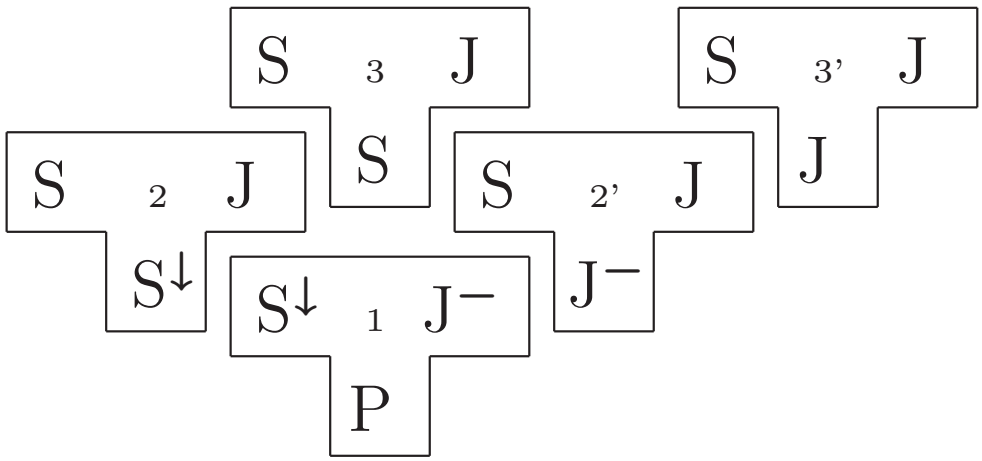
using basically our favourite language.

Combining the two compilers, we get:



which is an inefficient SCALA compiler (based on generated Java code) generating efficient Java code.

A final combination gives us what we want:



an efficient SCALA compiler, written in SCALA, running on the Java platform.