CSc 553

Principles of Compilation

0 : Administrivia

Department of Computer Science
University of Arizona

collberg@gmail.com

Introduction

Class	:	553 – Principles of Compilation
Lecturer	:	Christian Collberg
Email	:	collberg@gmail.com
WWW	:	http://www.cs.arizona.edu/∼collberg
Office	:	734
Office Hours	:	Tuesday 09:00-10:00
Phone	:	621-6612
Lectures	:	TTh 12:30-13:45, GLD-S 701

Prerequisites

- You should have had an introductory compiler class. Specifically, you should understand parsing and semantic analysis fairly well, and know something of code generation. This is normally taught in an undergraduate compiler class such as 453. If you lack this background you should read the relevant chapters in the textbook.
- You should be a proficient C programmer. Some knowledge of Java may also be necessary.
- Finally, you should have some background in an assembly language. It doesn't matter which one.

Compiler Design: We will...

- gain a deeper understanding of compiler design in general;
- gain a deeper understanding of programming language design;
- learn more about runtime systems including interpretation, garbage collection, exception handling;
- look more closely at code generation and code optimization;
- examine parallelizing compilers.

Topics

interpretation: Instruction sets, fast interpreter implementation.

fancy programming constructs: Exception handling, garbage collection, iterators, modules, object oriented language features.

code generation: Instruction selection, register allocation, instruction scheduling, peephole optimization.

program analysis: Alias and shape analysis, dependence analysis, data-flow analysis, control-flow analysis, type hierarchy analysis.

code optimization: Global optimization, memory hierarchy optimizations, inlining.

scientific code: FORTRAN, parallel computers, parallelizing compilers.

Syllabus

You are responsible for reading
and understanding this syllabus.
If you have any concerns or issues
about the information in this document
you should bring them up during the
first week of class.

Exam-schedule

1. There is no midterm exam.
2. The final exam is scheduled for Wed May 11, 1:00-3:00.

Coursework

Course Work and Assessment

Assignment 1 Interpretation.

Assignment 2 Garbage collection and Object-Orientation.

Assignment 3 Code generation.

Assignment 4 Optimization.

Presentation One 20-minute presentation of a research paper.

- The assignments will be done in teams of 2 students.
- Assessment: Assignments 50%%, Presentation 10%%, Final 40%%.

Presentation

- You will read one research paper related to compiler design, and
  1. prepare a 4-5 page summary handout (written in LaTeX),
  2. prepare a 15-25 slide presentation (written in LaTeX/Beamer),
  3. give a 20-minute presentation,
  4. prepare 3 (high-level) questions suitable for a final exam.
- I will give you LaTeX templates for the handouts and slides.
- The presentation and handout should
  1. clearly describe the problem the authors solve,
  2. give a high-level overview of the techniques employed,
  3. present the results of the paper (theorems, performance, tools, etc.),
  4. give your own opinion of the ideas, techniques, solutions, etc. presented in the paper, and suggestions for how they could be extended or improved.
- You may not plagiarize text/images from the paper.

Assessment Scheme

Tests, Quizzes, and Assignments

There will be

1. one comprehensive final exam, worth a total of 40%;
2. four programming assignments worth a total of 50%;
3. one paper presentation worth a total of 10%.

Late Assignments

- Assignments handed in no more than 24 hours late will incur a 10% penalty.
- Assignments handed in more than 24 but no more than 48 hours late will incur a 20% penalty.
- Assignments handed more than 48 hours after the deadline will receive a grade of 0.

Making up Tests

You cannot make up the final exam unless

1. you have notified the instructor in writing (email is fine) or by phone prior to the test that you will be absent, and
2. you receive permission from the instructor to take the test at a later date.

Curving

- All grades (for exams, quizzes, assignments, etc) will be curved up by throwing away the highest grade in the class and scaling up such that the second highest grade is 100.
- The curving is done to adjust for particularly difficult tests/assignments, and to prevent an outlier from skewing the grade distribution.
- You cannot, after scaling, receive more than 100 on any exam, quiz, assignment, etc.

Grade Assignment

- You will fail the class if you get less than 50 (after curving) on the final exam.
- Otherwise, a curved total grade of [90,100] gives you an A, [80,89] a B, [70,79] a C, [60,69] a D, and 59 and below an E.

Incomplete work policy

- Except under exceptional circumstances I will not assign incomplete grades.
- I decide what is an exceptional circumstance.

Detailed Grading Scheme

- To avoid any ambiguities, I have formalized the informal rules given above.
- The rules below should be considered *minimum* requirements to achieve a particular grade. The instructor reserves the right to do additional adjustments, as necessary.
- Any contradictions, omissions, errors, or ambiguities in the grading scheme will be resolved by the instructor.
- Any issues or concerns regarding the grading scheme should be brought to the attention of the instructor within the first week of class.

Details — Curving

- All raw scores range from 0 to 100.
- Each individual score (final, midterm, quizzes, assignments) will be curved using the function

$$\mathrm{curve}(\bar{x}, s) = \min(100, (100.0/\max(\bar{x} - \max(\bar{x})))\bar{x}_s)$$

  where $\bar{x}$ is a set of scores (for an assignment, a test, etc.) and s is a student.
- Note: $-$ is set subtraction.
- $\mathrm{curve}(\bar{x}, s)$ returns s's score, curved up by $100.0/\mathit{2nd\_highest\_class\_score}$.

Details — Curving...

- For example, assume the following final exam scores:

  34 45 66 88 98

  After the curve has been applied, the scores will be

  38.6 51.1 75 100 100

Details — Exams

final exam:

- Let $\bar{f}$ be the set of final exam scores.
- Let $\bar{f}^s$ be the final exam score for student s.
- Let $\mathcal{W}^f$ be the weight of the final exam (40%).
- $\bar{t}_f^s = \text{curve}(\bar{f}, s)\mathcal{W}^f$ is the curved final score for s.

Details — Presentation

- Let $\bar{p}$ be the set of presentation scores.
- Let $\bar{p}^s$ be the presentation score for student s.
- Let $\mathcal{W}^p$ be the weight of the presentation (10%).
- $\bar{t}_p^s = \text{curve}(\bar{p}, s)\mathcal{W}^p$ is the curved presentation score for s.

Details — Assignments

- Let $\bar{a}_i$ be the set of scores for the i:th assignment.
- Let $\bar{a}_i^s$ be the score for student s on the i:th assignment.
- Let $\mathcal{W}_i^a$ be the weight of the i:th assignment ($\sum_i \mathcal{W}_i^a = 50\%$).
- Let $\bar{\alpha}_i^s$ be the assignment score after late penalties have been applied:

$$\bar{\alpha}_i^s = \begin{cases} \bar{a}_i^s & \text{if the assignment is handed in on time} \\ 0.9\bar{a}_i^s & \text{if the assignment is } > 0 \text{ and } \leq 24 \text{ hours late} \\ 0.8\bar{a}_i^s & \text{if the assignment is } > 24 \text{ and } \leq 48 \text{ hours late} \\ 0 & \text{if the assignment is } > 48 \text{ hours late} \end{cases}$$

Details — Assignments. . .

- $\bar{t}_a^s = \sum_i (\text{curve}(\bar{\alpha}_i, s)\mathcal{W}_i^a)$ is the total curved assignment score for student s.
- If, for whatever reason, the actual number of assignments is less than the planned number, the $\mathcal{W}_i^a$'s will be scaled up uniformly.

Details — Total Scores

- The raw total score for student s is

$$\bar{t}_s = \bar{t}_f^s + \bar{t}_p^s + + \bar{t}_a^s$$

- For every absence > 2 I will subtract 4 points from the final grade:

$$\bar{u}_s = \bar{t}_s - (\max(0, \#\ of\ \ absences - 2)) * 4$$

- We round up to the nearest integer:

$$\text{total}_s = \lceil \bar{u}_s \rceil$$

Details — Grade Assignment

- The final grade assignment for student s is

$$\text{grade}_s = \begin{cases} E & \text{if } t_f^s < 50 \\ \begin{cases} A & \text{if } \text{total}_s \in [90, 100] \\ B & \text{if } \text{total}_s \in [80, 89] \\ C & \text{if } \text{total}_s \in [70, 79] \\ D & \text{if } \text{total}_s \in [60, 69] \\ E & \text{if } \text{total}_s < 60 \end{cases} & \text{otherwise} \end{cases}$$

- In other words, a student with a curved final exam score $t_f^s < 50$ will fail the class, regardless of their results on the other assessment categories.

Attendance Policy

- My goal is to keep class attendance high so that we can get good discussions going in the class.
- **You are allowed to miss 2 (two) classes during the semester without penalty. For every absence after that I will deduct 4 (four) points off your final grade.**

The Handouts

Handouts & Other Material I

1. The textbook is "Compilers – Principles, Techniques, and Tools", by Aho, Sethi, Ullman.
2. I always make copies of my transparencies available to students. Note that
   - I do this to relieve you of having to take notes during lectures,
   - they are not substitutes for reading the textbook,
   - their primary purpose is to remind you of what you need to study for the exam.

Handouts & Other Material II

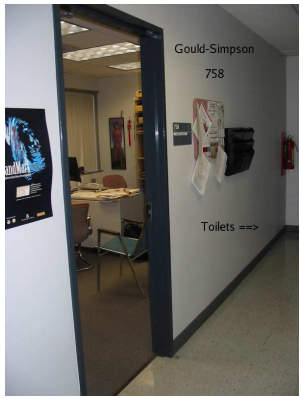
3. Various manuals and papers will be handed out during class. Extra copies can be picked up from the boxes outside my office.
4. Various information regarding the paper (including postscript files of the handouts) can be found on the info-bahn:

   http://www.cs.arizona.edu/~collberg/Teaching/553/2011/index.html

Policies

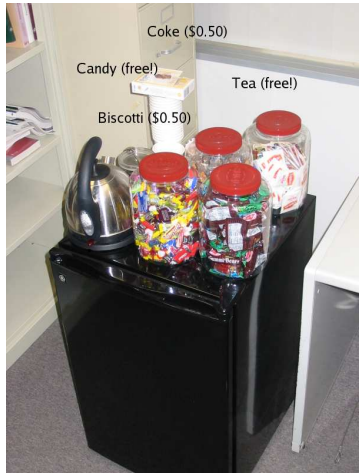
Office hours

- Office hours: Tuesday 09:00-10:00
- I use an open door policy:



Collberg's Café

- Please come and see me to chat, ask questions, or snack:



Subject to Change Policy

- The information contained in this course syllabus, other than the grade and absence policies, may be subject to change with reasonable advance notice, as deemed appropriate by the instructor.
- The instructor reserves the right to
  1. add, drop, or change topics;
  2. change exam or homework dates, etc.
- Changes will be announced in class and on the class web site!
  You are responsible for checking this site regularly.

Notification of Objectionable Materials

- There is no objectionable material in this class.

Students with Disabilities

- If you anticipate barriers related to the format or requirements of this course, please meet with me so that we can discuss ways to ensure your full participation in the course.
- If you determine that disability-related accommodations are necessary, please register with Disability Resources (621-3268; drc.arizona.edu) and notify me of your eligibility for reasonable accommodations. We can then plan how best to coordinate your accommodations.

Student Code of Academic Integrity

- Assignments in this course require individual attention and effort to be of any benefit. All work is expected to be that of each student alone. You may not consult with others, except in ways specifically authorized by the course instructor. You also may not plagiarize another person's work or copy another person's code.

Student Code of Academic Integrity. . .

- Students are responsible for understanding and complying with the University's Code of Academic Integrity. A synopsis of the Code is attached; the full text is available from the Office of the Dean of Students in Room 203 Old Main. Among other provisions, the Code demands that the work you submit is your own, and that graded papers and exams will not subsequently be tampered with. Copying of another student's programs or data, or writings is prohibited when they are part of a published class assignment; it is immaterial whether the copying is by computer, xerox, pen or other means. Witting collaboration in allowing such copying is also a Code violation.

Student Code of Academic Integrity. . .

- Assignments in this course require individual attention and effort
- Violations of the Code will, at minimum, result in loss of credit for a graded item. An egregious first violation or any second violation will minimally result in failure of the entire course.
- See also http://studpubs.web.arizona.edu/policies/cacaint.htm the University of Arizona Code of Academic Integrity.

I take academic integrity seriously! I will report *every* violation!

Expected classroom behavior

- Be courteous and treat others in the class with respect.
- Please be courteous to other students by refraining from talking, playing loud music in your headphones, etc.
- Silence cell phones, pagers, etc.
- We come to class to learn: don't read the newspaper, solve cross-word puzzles, etc.
- Treat the TAs with respect: they do their best to grade your assignments on time, help you with software installation problems, help you with assignments, etc. But they have their own class work to attend to, too.

Policies against threatening behavior

- Read and abide by the following link:
  http://policy.web.arizona.edu/~policy/threaten.shtml.

Now What?

Let's Have Fun!!![1] 

[1] That's right — compilers are fun!

```
From:  Dwight VandenBerghe, dwight@pentasoft.com
Date:  31 May 1997 09:43:49 -0400
Newsgroups:  comp.compilers
> why did we all get into compilers?
I was seventeen and had a lot of time on my hands.  I
was a programmer for the US Marines, stationed in Da
Nang, Vietnam, in the mid sixties, and during the
long nights I would go through the IBM microfiche -
they kept a full set in the computer room, all the
manuals and all the source code for all their tools.
I got interested in the COBOL compiler, and I read
through the assembler code for it.  I still remember
reading the instructions that scanned in an integer.
It just thrilled me, seeing how that magic was done.
I stayed up late into the early morning, trying to
figure it all out.
```

It's 3:10AM here now, and I've just finished downloading some papers from Norway on attribute grammars in ML. I have to go print them out. I'm dead tired, but this stuff is so exciting that sleep will have to wait. I'm 48 now, and my youngest boy is almost seventeen. My grandkids are great, and my wife is a gem, but you know, I hope that I am still cheating sleep for another couple of decades, because I feel seventeen, not nearly fifty. Compilers are magical. Compilers are like the two opposing mirrors in the funhouse - you see infinity, yet you are clearly finite. Writing a compiler, you go back and forth, from the infinite to the practical, over and over again.

And now, with the advent of functional programming into my life, there is an elegance to the code that I haven't seen before. The functional style seems to me to be made for us compiler jockeys. And the advances that have been made in this field: BURS theory, the SUIF system, transformational tree rewrite systems, higher-order attribute grammars, the exciting new work with LL(infinite) parser generators ... it's simply thrilling. Nothing else compares, at least, for me. Applications suck; device drivers are tedious; operating systems and file management and databases are boring. After 31 years, I've pretty much done it all, and if it wasn't for compilers I'd be bored stiff. Yet here it is, the wee hours yet again. Last night I finally hauled myself to sleep at a little after 2AM, because I found some great papers on polymorphic type theory on some server in the UK. So it goes.
Dwight

What's available in Compiler Jobs?

From: Clifford Click, cliff.click@Eng.Sun.COM
Newsgroups: comp.compilers

> 2. What do you think is the future for this market?
> 3. Is the demand growing?

Skilled *optimizing* compiler writers are in high demand. The market is small, but the the labor pool is even smaller. (The demand for people who make functioning parsers is much smaller, probably due to the ease of using lex & yacc).

The future for this market? Welll... as long as chip architects and language designers are alive I got a job. Merced looks like full time employment for 20+ people for 5+ years (OUTSIDE of Intel & HP; they probably have double that number already and are always looking for more). [Again, I'm talking about optimizing compiler writers; big optimizers grow big support groups that hire more people in nearby fields]
And yes, I believe the demand is growing.

www.compilerjobs.com/jobs/microsoft.htm

Job Title	Software Design Engineer Lead
Description	The Visual Basic.Net team is in search of an exceptionally strong development lead, experienced and interested in driving the development effort on Visual Basic .Net compiler. Responsibilities include leading a team in designing and implementing new VB.Net language features. This is a great opportunity to be involved with the development of the most successful development tool and one of the most important technologies at Microsoft.
Qualifications	Solid knowledge of C/C++ and multi-threaded programming. A BA/BS or MS degree in CS.
Website	www.microsoft.com

www.compilerjobs.com/jobs/cray.htm

Job Location	Mendota Heights, MN USA
Description	Cray Inc. designs, builds, and sells high-performance MPP, vector processor and scalable shared memory parallel computer systems.
Qualifications	Proficient in C, C++, and Fortran. Minimum of 2-3 years experience in writing compilers. Familiar with Unix. Experience is desired in: Compiling for Multiprocessors, Parallel programming, Performance tuning.
Education:	B.S. in Computer Science or equivalent experience.
Our website	http://www.cray.com

www.compilerjobs.com/jobs/cloakware.htm

Job Title	Compiler Designer - Permanent
Job Location	Ottawa, Ontario, CANADA
Description	Working with our core technology team you will develop compilers for our Tamper-Resistent Software (TRS).
Our website	http://www.cloakware.com
Qualifications	Cloakware is seeking expert Compiler Writers, with knowledge any or a combination of the following: Optimization, Program Transformation, Combinatorics, Obfuscation Techniques, Program slicing Techniques, Numerical Analysis, Semetric Representations for Control- and Data-Flow, Control- and Data-Flow Dependencies

www.compilerjobs.com/jobs/irvine.htm

Job Title	Post-Doctoral Researcher
Description	Several postdoctoral research positions are available, working on new approaches to mobile code; this is compiler-related research.
Qualifications	Applicants must have attained a Ph.D. in Computer Science or related field and have prior experience with software systems, possess demonstrated familiarity with code generation for modern RISC architectures, and the Ada and Java programming languages.
Website	www.ics.uci.edu/~franz