

Principles of Compiler Design

Sanjeev K Aggarwal
Department of Computer Science
and Engineering, IIT Kanpur
ska@iitk.ac.in

1

Motivation for doing this course

- Language processing is an important component of programming
- A large number of systems software and application programs require structured input
- Operating Systems (command line processing)
- Databases (Query language processing)
- Type setting systems like Latex, Nroff, Troff, Equation editors, M4
- VLSI design and testing,
- Software quality assurance and software testing

2

- XML, html based systems, Awk, Sed , Emacs , vi ...
- Form processing, extracting information automatically from forms
- Compilers, assemblers and linkers
- High level language to language translators
- Natural language processing
- Where ever input has a structure one can think of language processing
- Why study compilers?
Compilers use the whole spectrum of language processing technology

3

Many common applications require structured input during development phase (design of banner programme of Unix)

```
xxxxxxxxx      9x
xxxxxxxxx      9x
xxxxxxxxx      9x
   xxx         3b 3x      3 9x
   xxx         3b 3x      6 3b 3x
   xxx         3b 3x      3 9x
   xxx         3b 3x
   xxx         3b 3x
   xxx         3b 3x
xxxxxxxxx      9x
xxxxxxxxx      9x
xxxxxxxxx      9x
```

4

What will we learn in the course?

- How high level languages are implemented to generate machine code. Complete structure of compilers and how various parts are composed together to get a compiler
- Course has theoretical and practical components. Both are needed in implementing programming languages. The focus will be on practical application of the theory.
- Emphasis will be on algorithms and data structures rather than proofs of correctness of algorithms.
- Theory of lexical analysis, parsing, type checking, runtime system, code generation, optimization (without going too deep into the proofs etc.)
- Techniques for developing lexical analyzers, parsers, type checkers, run time systems, code generator, optimization. Use of tools and specifications for developing various parts of compilers

5

What do we expect to achieve by the end of the course?

- Students are confident that they can use language processing technology for various software developments
- Students are confident that they can design, develop, understand, modify/enhance, and maintain compilers for (even complex!) programming languages

6

Organization of the course

- Approximately 40 lectures of one hour each
- One large programming project to be done in group of four students. The project will have 4-5 parts and will be evaluated every 2-3 weeks during the semester. It is important to start early. (25% credit)
 - Copying old code is cheating. **Do not Cheat.**
 - Late submissions will be increasingly penalized.
 - Submissions must happen by noon on the due day
- Some small programming assignments and home work (10% credit)
 - Do not copy. However, you are free to discuss
 - Late submissions will be increasingly penalized.
 - Submissions must happen by noon on the due day
- Two mid semester examinations (15% + 15% credit)
- One end semester examination (30% credit)
- Participation in the discussions in the class + class notes (5% credit)

7

Class Notes

- Groups of three students to be formed by the instructor in round robin fashion
- Each group will be responsible for preparing notes (both writing and editing) for two consecutive lectures
- Notes should be related to slides or a group of slides
- Notes should be added to the power point slides of the lectures
- Slides and notes will be made available to the students
- Deadline for notes: **Seven days from the second lecture**

8

Required Background and self reading

- This is a 5 unit course and, therefore, will have about 25% more load than other professional courses. You should be prepared for more work and long hours of programming
- Courses in data structures, computer organization, operating systems
- Proficiency in C/C++/Java programming languages
- Knowledge of at least one assembly language, assembler, linker & loader, symbolic debugger
- **You are expected to read the complete book (except the chapter on code optimization) on Compiler Design by Aho, Sethi and Ullman. All the material will not be covered in the lectures**

9

Things to do immediately

- Form groups of 4 students
- Inform the instructor
 - Send an email with subject line **"CS335 group"**
 - Give roll number, name and email ids of all the group members in the message
 - **Deadline Dec 31, 2004**
- **Give feed back through out the semester: either personally or through email or anonymous.**
PLEASE HELP ME IMPROVING THE SLIDES (both presentation style and the material), AND THE NOTES!!
- **First assignment** Write a simulator for the stack machine described in the text book by Aho, Sethi and Ullman in section 2.8. Submission date: **Jan 10, 2005**

10

Bit of History

- How are programming languages implemented? Two major strategies:
 - Interpreters (old and much less studied)
 - Compilers (very well understood with mathematical foundations)
- Some environments provide both interpreter and compiler. Lisp, scheme etc. provide
 - Interpreter for development
 - Compiler for deployment
- Java
 - Java compiler: Java to interpretable bytecode
 - Java JIT: bytecode to executable image

11

Some early machines and implementations

- IBM developed 704 in 1954. All programming was done in assembly language. Cost of software development far exceeded cost of hardware. Low productivity.
- Speedcoding interpreter: programs ran about 10 times slower than hand written assembly code
- John Backus (in 1954): Proposed a program that translated high level expressions into native machine code. Skepticism all around. Most people thought it was impossible
- Fortran I project (1954-1957): The first compiler was released

12

Fortran I

- The first compiler had a huge impact on the programming languages and computer science. The whole new field of compiler design was started
- More than half the programmers were using Fortran by 1958
- The development time was cut down to half
- Led to enormous amount of theoretical work (lexical analysis, parsing, optimization, structured programming, code generation, error recovery etc.)
- Modern compilers preserve the basic structure of the Fortran I compiler !!!

13

References

- Compilers: Principles, Tools and Techniques by Aho, Sethi and Ullman
soon to be replaced by "21st Century Compilers" by Aho, Sethi, Ullman, and Lam
- Crafting a Compiler in C by Fischer and LeBlanc
soon to be replaced by "Crafting a Compiler" by Fischer
- Compiler Design in C by Holub
- Programming Language Pragmatics by Scott
- Engineering a Compiler by Cooper and Torczon
- Modern Compiler Implementation (in C and in Java) by Appel
- Writing Compilers and Interpreters by Mak

14

- Compiler Design by Wilhelm and Maurer
- A Retargetable Compiler: Design and Implementation by Fraser and Hanson
- Compiler Construction by Wirth
- The Theory of Parsing, Translation and Compiling (vol 1 & 2) by Aho and Ullman (old classic)
- Introduction to Compiler Construction with Unix by Schreiner and Friedman
- Compiler Design and Construction: Tools and Techniques by Pyster
- Engineering a Compiler by Anklam and Cutler
- Object Oriented Compiler Construction by Holmes
- The Compiler Design Handbook edited by Srikant and Shankar

15