CS 314 Principles of Programming Languages
Lecture 1: Overview and Basics

Zheng (Eddy) Zhang
Rutgers University
January 17th, 2018
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 (C++), parallel (OpenMP/CUDA)
  ‣ 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
  ‣ To understand the implementation challenges of different programming constructs/features

Programming languages are tools ⇒ Understand how to design or use them
Course Information

Prerequisites:
- CS 205 (Introduction to Discrete Structures)
- CS 211 (Computer Architecture)

Important facts:
- Staff: Prof. Zheng Zhang
- Lectures: Monday/Wednesday 3:20pm - 4:40pm
- Recitations:
  - Section 5, Monday 8:25pm - 9:20pm LSH-B267 LIV
  - Section 6, Wednesday 10:35am - 11:30am LSH-B115 LIV
  - Section 7, Monday 12:15pm - 1:10pm, TIL-242

Basis for grades (subject to changes):
- 10% homework
- 25% mid-term exam
- 35% final exam (cumulative)
- 30% three major programming projects
- 5% extra-credit homework and project questions
Course Information

Textbook:
• Required:
• Suggested:

Course website:
• All lectures will be posted to our course website.
• The course website will be up soon. A previous version: https://www.cs.rutgers.edu/courses/314/classes/fall_2016_zhang/

Online discussion:
• There will be a news group on Sakai.
  All questions regarding homework and projects should be posted here.
  Please DO NOT post solutions.

All programming will be done on the ilab cluster. Get yourself an ilab account (see link on bottom of our 314 website) if you don’t have one.
Course Information

**Academic Integrity** (see our web page)
- read-protect your directories and files (ilab)
- no group projects
- will use software tools to check code plagiarism

14 WEEKS, no “make-up” work after the end of the course.

**IMPORTANT INFORMATION** will be posted on course website and Sakai
- Grading of homework and projects
- Instructions of how to submit programming projects
- Partial credit for **justified** late project submissions

**Email TAs and me:**
- Subject line has to start with 314
  E.g., 314: Question about my midterm exam
- No project and homework questions; post them on Sakai discussion forum.
Course Information

Special permission numbers:
• Currently all three sections are full
• Will try to accommodate as many students as I can if there is vacancy
• Talk to me after class
Course Information

I use detailed lecture notes
- I try to moderate my speed
- You are encouraged to ask questions
- All lecture notes are on the Web (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 strongly recommend coming to the lectures
What is the Purpose of a Programming Language?

What is a language for?

• A way of thinking
  
  *From the user's point of view:*
  
  A way of expressing and reasoning about algorithms.

• An abstraction of virtual machine
  
  *From the implementer's point of view:*
  
  A way of specifying what you want the hardware to do without getting down into the bit.
This is a C program that uses two one-dimensional arrays `a` and `b` of size `SIZE`. The arrays are initialized, and then a sum reduction is performed. The size of the arrays and the result of the sum reduction is printed out.

```c
#include <stdio.h>
#define SIZE 100
int main() {
    int a[SIZE], b[SIZE];
    int i, sum;
    for (i=0; i<SIZE; i++) {
        a[i] = 1;
        b[i] = 2;
    }
    sum = 0;
    for (i=0; i<SIZE; i++)
        sum = sum + a[i] + b[i];
    printf("for two arrays of size %d, sum = %d\n", SIZE, sum);
}
```

14 Lines in Total
Why Use Anything Besides Machine Code?

Compiler: gcc -O3 -S example.c —> example.s

```assembly
.file "example.c"
.version "01.01"
gcc2_compiled.:
.section .rodata.str1.32,"aMS",@progbits,1
.align 32
.LC0:
.string "for two arrays of size %d, sum = %d\n" .text
.align 4
.globl main
.type main,@function
main:
pushl %ebp
movl %esp, %ebp
xorl %eax, %eax
subl $808, %esp
movl $99, %edx
.p2align 2
.L21:
movl $1, -408(%ebp,%eax)
movl $2, -808(%ebp,%eax)
addl $4, %eax
dcl %edx
jns .L21
xorl %ecx, %ecx
xorl %eax, %eax
movl $99, %edx
.p2align 2
```
addl -408(%ebp,%eax), %ecx
addl -808(%ebp,%eax), %ecx
addl $4, %eax
decl %edx
jns .L26
pushl %eax
pushl %ecx
pushl %edx
pushl $100
pushl $.LC0
call printf
addl $16, %esp
leave
ret
.Lfe1:
.size main,.Lfe1-main
.ident "GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-112)"

Results of “Compiler: gcc -O3 -S example.c —> example.s”
45 Lines in Total
Why Use Anything Besides Machine Code?

gcc -o example.o -O3 example.c;
strip example.o; objdump -d example.o;

objdump: example.o: No symbols

eexample.o: file format elf32-sparc

Disassembly of section .text:

00010444 <.text>:
  10444: bc 10 20 00 clr %fp
  10448: e0 03 a0 40 ld [ %sp + 0x40 ], %l0
  1044c: a2 03 a0 44 add %sp, 0x44, %l1
  10450: 9c 23 a0 20 sub %sp, 0x20, %sp
  10454: 80 90 00 01 tst %g1
  10458: 02 80 00 04 be 0x10468
  1045c: 90 10 00 01 mov %g1, %o0
  10460: 40 00 40 c4 call 0x20770
  10464: 01 00 00 00 nop
  10468: 11 00 00 41 sethi %hi(0x10400), %o0
  1046c: 90 12 22 d8 or %o0, 0x2d8, %o0 ! 0x106d8
  10470: 40 00 40 c0 call 0x20770
  10474: 01 00 00 00 nop
  10478: 40 00 00 91 call 0x106bc
  1047c: 01 00 00 00 nop
  10480: 90 10 00 10 mov %l0, %o0
  10484: 92 10 00 11 mov %l1, %o1
  10488: 95 2c 20 02 sll %l0, 2, %o2
  1048c: 94 02 a0 04 add %o2, 4, %o2
  10490: 94 04 40 0a add %l1, %o2, %o2
  10494: 17 00 00 82 sethi %hi(0x20800), %o3
10498: 96 12 e0 a8  or  %o3, 0xa8, %o3  ! 0x208a8
1049c: d4 22 c0 00  st  %o2, [ %o3 ]
104a0: 40 00 00 4e  call  0x105d8
104a4: 01 00 00 00  nop
104a8: 40 00 40 b5  call  0x2077c
104ac: 01 00 00 00  nop
104b0: 40 00 40 b6  call  0x20788
104b4: 01 00 00 00  nop
104b8: 81 c3 e0 08  retl
104bc: ae 03 c0 17  add  %o7, %l7, %l7
104c0: 9d e3 bf 90  save  %sp, -112, %sp
104c4: 11 00 00 00  sethi  %hi(0), %o0
104c8: 2f 00 00 40  sethi  %hi(0x10000), %l7
104cc: 7f ff ff fb  call  0x104b8
104d0: ae 05 e2 54  add  %l7, 0x254, %l7  ! 0x10254
104d4: 90 12 20 0c  or  %o0, 0xc, %o0
104d8: d2 05 c0 08  ld  [%l7 + %o0 ], %o1
104dc: d4 02 40 00  ld  [%o1 ], %o2
104e0: 80 a2 a0 00  cmp  %o2, 0
104e4: 12 80 00 23  bne  0x10570
104e8: 11 00 00 00  sethi  %hi(0), %o0
104ec: 90 12 20 10  or  %o0, 0x10, %o0  ! 0x10
104f0: d4 05 c0 08  ld  [%l7 + %o0 ], %o2
104f4: d2 02 80 00  ld  [%o2 ], %o1
104f8: d0 02 40 00  ld  [%o1 ], %o0
104fc: 80 a2 20 00  cmp  %o0, 0
10500: 02 80 00 0f  be  0x1053c
10504: 11 00 00 00  sethi  %hi(0), %o0
10508: a0 10 00 0a  mov  %o2, %l0
1050c: d0 04 00 00  ld  [%l0 ], %o0
10510: 90 02 20 04  add  %o0, 4, %o0
10514: d0 24 00 00  st  %o0, [ %l0 ]
10518: d2 02 3f fc  ld [ %o0 + -4 ], %o1
1051c: 9f c2 40 00  call %o1
10520: 01 00 00 00  nop
10524: d0 04 00 00  ld [ %l0 ], %o0
10528: d2 02 00 00  ld [ %o0 ], %o1
1052c: 80 a2 60 00  cmp %o1, 0
10530: 12 bf ff f9  bne 0x10514
10534: 90 02 20 04  add %o0, 4, %o0
10538: 11 00 00 00  sethi %hi(0), %o0
1053c: 90 12 20 1c  or %o0, 0x1c, %o0 ! 0x1c
10540: d2 05 c0 08  ld [ %l7 + %o0 ], %o1
10544: 80 a2 60 00  cmp %o1, 0
10548: 02 80 00 05  be 0x1055c
1054c: 13 00 00 00  sethi %hi(0), %o1
10550: 92 12 60 08  or %o1, 8, %o1 ! 0x8
10554: 40 00 40 90  call 0x20794
10558: d0 05 c0 09  ld [ %l7 + %o1 ], %o0
1055c: 11 00 00 00  sethi %hi(0), %o0
10560: 90 12 20 0c  or %o0, 0xc, %o0 ! 0xc
10564: d4 05 c0 08  ld [ %l7 + %o0 ], %o2
10568: 92 10 20 01  mov 1, %o1
1056c: d2 22 80 00  st %o1, [ %o2 ]
10570: 81 c7 e0 08  ret
10574: 81 e8 00 00  restore
10578: 9d e3 bf 90  save %sp, -112, %sp
1057c: 81 c7 e0 08  ret
10580: 81 e8 00 00  restore
10584: 9d e3 bf 90  save %sp, -112, %sp
10588: 11 00 00 00  sethi %hi(0), %o0
1058c: 2f 00 00 40  sethi %hi(0x10000), %l7
10590: 7f ff ff ca  call 0x104b8
10594: ae 05 e1 90  add %l7, 0x190, %l7 ! 0x10190
10598: 90 12 20 18  or %o0, 0x18, %o0
1059c: d2 05 c0 08 ld [%l7 + %o0], %o1
105a0: 80 a2 60 00 cmp %o1, 0
105a4: 02 80 00 08 be 0x105c4
105a8: 13 00 00 00 sethi %hi(0), %o1
105ac: 92 12 60 08 or %o1, 8, %o1 ! 0x8
105b0: 15 00 00 00 sethi %hi(0), %o2
105b4: d0 05 c0 09 ld [%l7 + %o1], %o0
105b8: 94 12 a0 04 or %o2, 4, %o2
105bc: 40 00 40 79 call 0x207a0
105c0: d2 05 c0 0a ld [%l7 + %o2], %o1
105c4: 81 c7 e0 08 ret
105c8: 81 e8 00 00 restore
105cc: 9d e3 bf 90 save %sp, -112, %sp
105d0: 81 c7 e0 08 ret
105d4: 81 e8 00 00 restore
105d8: 9d e3 bc 70 save %sp, -912, %sp
105dc: 92 07 be 60 add %fp, -416, %o1
105e0: 94 07 bc d0 add %fp, -816, %o2
105e4: 86 10 00 09 mov %o1, %g3
105e8: 84 10 00 0a mov %o2, %g2
105ec: 9a 10 20 01 mov 1, %o5
105f0: 98 10 20 02 mov 2, %o4
105f4: 90 10 20 00 clr %o0
105f8: 96 10 20 63 mov 0x63, %o3
105fc: da 22 00 03 st %o5, [ %o0 + %g3 ]
10600: d8 22 00 02 st %o4, [ %o0 + %g2 ]
10604: 96 82 ff ff addcc %o3, -1, %o3
10608: 1c bf ff fd bpos 0x105fc
1060c: 90 02 20 04 add %o0, 4, %o0
10610: 9a 10 00 0a mov %o2, %o5
10614: 84 10 00 09 mov %o1, %g2
10618: 94 10 20 00 clr %o2
1061c: 98 10 20 00 clr %o4
10620: 96 10 20 63 mov 0x63, %o3
10624: d0 03 00 02  ld [ %o4 + %g2 ], %o0
10628: 96 82 ff ff  addcc %o3, -1, %o3
1062c: d2 03 00 0d  ld [ %o4 + %o5 ], %o1
10630: 90 02 80 08  add %o2, %o0, %o0
10634: 94 02 00 09  add %o0, %o1, %o2
10638: 1c bf ff fb  bpos 0x10624
1063c: 98 03 20 04  add %o4, 4, %o4
10640: 11 00 00 41  sethi %hi(0x10400), %o0
10644: 90 12 22 f8  or %o0, 0x2f8, %o0 ! 0x106f8
10648: 40 00 40 59  call 0x207ac
1064c: 92 10 20 64  mov 0x64, %o1
10650: 81 c7 e0 08  ret
10654: 81 e8 00 00  restore
10658: 81 c3 e0 08  retl
1065c: ae 03 c0 17  add %o7, %l7, %l7
10660: 9d e3 bf 90  save %sp, -112, %sp
10664: 11 00 00 00  sethi %hi(0), %o0
10668: 2f 00 00 40  sethi %hi(0x10000), %l7
1066c: 7f ff ff fb  call 0x10658
10670: ae 05 e0 b4  add %l7, 0xb4, %l7 ! 0x100b4
10674: 90 12 20 14  or %o0, 0x14, %o0
10678: d2 05 c0 08  ld [ %l7 + %o0 ], %o1
1067c: d4 02 7f fc  ld [ %o1 + -4 ], %o2
10680: 80 a2 bf ff  cmp %o2, -1
10684: 02 80 00 09  be 0x106a8
10688: a0 02 7f fc  add %o1, -4, %l0
1068c: d0 04 00 00  ld [ %l0 ], %o0
10690: 9f c2 00 00  call %o0
10694: a0 04 3f fc  add %l0, -4, %l0
10698: d0 04 00 00  ld [ %l0 ], %o0
1069c: 80 a2 3f ff  cmp %o0, -1
106a0: 12 bf ff fb  bne 0x1068c
Disassembly of section .init:

000106bc <.init>:
  06bc: 9d e3 bf a0  save  %sp, -96, %sp
  06c0: 7f ff ff b1  call  0x10584
  06c4: 01 00 00 00  nop
  06c8: 7f ff ff e6  call  0x10660
  06cc: 01 00 00 00  nop
  06d0: 81 c7 e0 08  ret
  06d4: 81 e8 00 00  restore

Disassembly of section .fini:

000106d8 <.fini>:
  06d8: 9d e3 bf a0  save  %sp, -96, %sp
  06dc: 7f ff ff 79  call  0x104c0
  06e0: 01 00 00 00  nop
  06e4: 81 c7 e0 08  ret
  06e8: 81 e8 00 00  restore

Disassembly of section .plt:

00020740 <.plt>:

  20770: 03 00 00 30  sethi  %hi(0xc000), %g1
  20774: 30 bf ff f3  b,a  0x20740
  20778: 01 00 00 00  nop
  2077c: 03 00 00 3c  sethi  %hi(0xf000), %g1
  20780: 30 bf ff f0  b,a  0x20740
  20784: 01 00 00 00  nop
Results of applying:
gcc -o example.o -O3 example.c;
strip example.o; objdump -d example.o;

207 Lines in Total
Why Use Anything Besides Machine Code?

Need for high-level programming languages for

- Readable, familiar notations
- Machine independence (portability)
- Consistency checks during implementation
- Dealing with scale
  
The art of programming is the art of organizing complexity.
  - Dijkstra, 1972

However,

- Acceptable loss of efficiency
  
  First FORTRAN compiler built by IBM, in 1957, translated into code as efficient as hand-coded code.
  - John Backus
Why Learn More than One Programming Language?

Need for high-level programming languages for

• Each language encourages thinking about a problem in a particular way.
• Each language provides different expressiveness & efficiency.

⇒ The language should match the problem.
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.
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 *Control Flow*
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*
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*
Parallel:

*Computations and data accesses happen concurrently*

- Task and data parallelism
- Different granularities: instruction, loop, or task level
- Synchronization: locks, message passing, . . .
- Key notions: *control* and *data dependencies*
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 Compilers

Implications:
- Intermediate representation (IR)
- Front end maps legal code into IR
- Back end maps IR onto target machine
- Simplify retargeting
- Allows multiple front ends and back ends

Front end is $O(n)$
Back end is NP-Complete
Syntax & semantic analyzer, IR code generator

Front End Responsibilities:
• Recognize legal programs
• Report errors
• Produce IR
• Preliminary storage map
• Shape the code for the back end

Much of front end construction can be automated
Scanner

- Maps characters into tokens — the basic unit of syntax
  
  \[
  x = x + y;
  \]
  
  becomes
  
  \[
  <\text{id}, x> = <\text{id}, x> + <\text{id}, y> ;
  \]

- 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
  
  ⇒ use specialized recognizer (lex)
**Parser**

- Recognize context-free syntax (Context Free Grammars)
- Guide context-sensitive analysis
- Construct IR(s)
- Produce meaningful error messages
- Attempt error correction

*Parser generators mechanize much of the work*
Compilers often use an abstract syntax tree.

\[
\begin{align*}
\text{\texttt{x + 2 - y}}
\end{align*}
\]

\[
\begin{array}{c}
+ \\
- \\
\text{<id, x>} \\
\text{<num, 2>} \\
\text{<id, y>}
\end{array}
\]
Back End

**Responsibilities**

- Translate IR into target machine code
- Choose instructions for each IR operation
- Decide what to keep in registers at each point
Modern optimizers are usually built as a set of passes. 
**Pass: reading and writing entire program**

**Typical passes**
- Discover & propagate constant values
- Reduction of operator strength
- Common subexpression elimination
- Redundant computation elimination
- Move computation to less frequently executed place (e.g., out of loops)
Things to do for next lecture:
  • Read Scott: Chapter 1 (covers today’s lecture)
  • Read Scott: Chapters 2.1 and 2.2; ALSU: Chapters 3.1 - 3.4
  • Get an ilab account
  • Learn to use Sakai discussion group

Recitations will start **Next Week**.