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), parallel (OpenMP, CUDA), quantum computing (Qiskit) and reversible computing (Janus)
  - 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 (summary):

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

Important facts:

staff: 
- Prof. Ulrich Kremer (instructor)
- Chun Leung Lau (TA)
- Xinyu Zhang (TA)
- TBD (graders)

lectures: 
Tue/Fri: 12:10pm-1:30pm, HLL-114

recitations: 
different times on Tuesdays
- section 1, 7:45pm - 8:40pm, SEC-202
- section 2, 5:55pm - 6:50pm, SEC-203
- section 3, 4:05pm - 5:00pm, SEC-205
- section 4, 2:15pm - 3:10pm, SEC-205

Basis for grades (subject to changes):

10% homework
2 × 30% two midterms exams and one final exam (best two out of three)
30% three major programming projects

Overall course grade will only be partially curved: You are guaranteed to get an A when above 90%, B above 80%, C above 70%, D above 60%. However, curve may push these boundaries lower (e.g.: A for 89%)
Course Information (Cont.)


- Additional (recommended) texts: see course web page

Course material is available on our class website at

www.cs.rutgers.edu/courses/314/classes/spring_2023_kremer

In addition, there is a piazza group (not yet set up) All questions regarding homeworks and projects MUST be posted here. You should read our piazza and cavas site at least once every other day. Homework solutions and grades will be posted on our canvas site.

Programming projects will be done on the ilab cluster. Get yourself an ilab account (see our 314 website). Learn to do the normal things in Linux — edit, compile, …
Course Information (Cont.)

**Academic Integrity** (see our web page)
→ read-protect your directories and files (ilab)
→ no group projects
→ will use MOSS for detecting software plagiarism

14 weeks, no “make-up” work after the end of the course. If there is a problem, let me know immediately.

**IMPORTANT INFORMATION/ANNOUNCEMENTS**
⇒ will be posted on canvas:

- Homeworks and projects, and their grades
- Instructions of how to submit programming projects
- Partial credit for late project submissions
- Sample solutions for homeworks
- Grades

Email TAs or me:

- **Subject line** has to start with 314:
  e.g., 314: Question about my midterm exam
- **No** project and homework questions; post them on the piazza discussion forums;
Course Information (Cont.)

Special permission numbers

https://www.cs.rutgers.edu/academics/undergraduate/course-registration-and-special-permission

Nearly everyone who submitted request and is class of 2023 or 2024 got an SPN (41 SPNs will be send out soon). Check your emails. Assigned SPNs “expire” after 5 days.

- No pre-requisite override
- We are currently at recitation room capacity (60 seats)
- Last day to add classes: Thursday, January 26

To get into the class, or change a section, please use the above special permission website. Some students will drop out, or will not use their SPNs.
I use lecture notes

- I try to moderate my speed
- You need to say STOP!
- All lecture notes are on the Web (PDF)
  - draft will be available before class, e.g., lec1.pdf
  - final version will have a mod suffix, e.g., lec1mod.pdf
- You should still take some notes, since not everything we will talk about in class will be in the notes, for instance examples.

I’ll tell you where we are in the book

- I don’t lecture directly from the book
- You need to read the book
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 structures
- communicating algorithms among people
- establishing correctness (allow reasoning)
- but also: provide foundation for different notions of computation

New challenge: How will new AI technology such as ChatGPT change the way we program (Matt Welsh article January 2023 article “End of Programming” in Communications of the ACM).
Why Use Anything Besides Machine Code?

This is a C program that uses two one-dimensional arrays \texttt{a} and \texttt{b} of size \texttt{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.

\begin{verbatim}
example.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);
}
\end{verbatim}
Why Use Anything Besides Machine Code?

Compiler: gcc -O3 -S example.c ⇒ example.s

.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
global main
type main,@function
main:
pushl %ebp
movl %esp, %ebp
xorl %eax, %eax
subl $808, %esp
movl $99, %edx
.align 2
.L21:
movl $1, -408(%ebp,%eax)
movl $2, -808(%ebp,%eax)
addl $4, %eax
decl %edx
jns .L21
xorl %ecx, %ecx
xorl %eax, %eax
movl $99, %edx
.align 2
.L26:
addl -408(%ebp,%eax), %ecx
addl -808(%ebp,%eax), %ecx
addl $4, %eax
decl %edx
jns .L26
pushl %eax
pushl %ecx
pushl $100
pushl $\text{.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)"
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

example.o: file format elf32-sparc

Disassembly of section .text:

00010444 <text>:
10444: bc 10 20 00 clr Xfp
10448: a0 03 a0 40 ld [ Xsp + 0x40 ], X10
1044c: a2 03 a0 44 add Xsp, 0x44, X11
10450: 9c 23 a0 20 sub Xsp, 0x20, Xsp
10454: 80 90 00 01 tst Xsp
10458: 02 80 00 04 be 0x10468
1045c: 90 10 00 01 mov Xg1, X0
10460: 40 00 00 04 call 0x20770
10464: 01 00 00 00 nop
10468: 11 00 00 41 sethi Xhi(0x10400), X0
1046c: 90 12 22 48 or Xlo, 0x248, X0 ! 0x106b8
10470: 40 00 00 0c 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 Xlo, X0
10484: 92 10 00 11 mov Xl1, X1
10488: 95 2c 20 02 all Xl0, 2, X2
1048c: 94 02 a0 4d add Xl2, 4, X3
10490: 94 04 40 0a add Xl1, X0, X2
10494: 17 00 00 82 sethi Xhi(0x20800), X3
10498: 96 12 e0 48 or Xlo, 0x848, X3 ! 0x20a88
1049c: 44 22 c0 00 st Xo2, [ Xo3 ]
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 ret1
104bc: ae 03 c0 17 add Xo7, Xl7, X17
104c0: 9d e3 bf 90 save Xsp, -112, Xsp
104c4: 11 00 00 00 sethi Xhi(0), X0
104c8: 2f 00 00 40 sethi Xhi(0x10000), X17
104cc: 7f ff ff fb call 0x104b8
104d0: ae 05 e2 54 add Xl7, 0x254, Xl7
104d4: d2 02 40 00 ld [ Xl7 + Xo0 ], Xo1
104d8: a0 82 a0 00 cmp Xo2, 0
104dc: 81 c3 e0 08 ret1
104e0: 9d e3 bf 90 save Xsp, -112, Xsp
104e4: 11 00 00 00 sethi Xhi(0), X0
104e8: 94 02 a0 4d add Xl2, 4, X3
104ec: 92 10 00 11 mov Xl1, X1
104f0: 95 2c 20 02 all Xl0, 2, X2
104f4: 94 02 a0 4d add Xl2, 4, X3
104f8: 94 04 40 0a add Xl1, X0, X2
104fc: d4 22 c0 00 st Xo2, [ Xo3 ]
10500: 40 00 00 4e call 0x105d8
10504: 01 00 00 00 nop
10508: 90 10 00 10 mov Xlo, X0
1050c: 82 42 e0 08 cmp Xo2, 0
10510: 90 02 20 04 add Xlo, 0x4, X0
10514: 92 20 00 00 st Xlo, [ X10 ]
10518: d2 02 3f fc ld [ Xo0 + -4 ], Xo1
1051c: 8e c2 40 00 call Xo1
10520: 01 00 00 00 nop
10524: 9b c3 e0 08 cmp Xo0
10528: d2 02 00 00 ld [ Xo0 ], Xo1
1052c: 80 a2 60 00 cmp Xo0, 0
10530: d0 02 20 04 add Xlo, 0x4, X0
10534: b0 04 00 00 msv Xo2, X10
10538: a0 44 00 00 ld [ Xlo ], Xo0
1053c: 90 12 20 10 or Xlo, 0x10, X0 ! 0x10
10540: d4 05 c0 08 ld [ Xl7 + Xo0 ], Xo2
10544: d2 02 80 00 ld [ Xo2 ], Xo1
10548: d0 02 40 00 ld [ Xo1 ], Xo0
1054c: 80 a2 20 00 cmp Xo0, 0
10550: 02 80 00 0f be 0x1053c
10554: 11 00 00 00 sethi Xhi(0), X0
10558: a0 10 00 0a msv Xo2, X10
1055c: a0 04 00 00 ld [ Xlo ], Xo0
10560: 90 02 20 04 add Xlo, 0x4, X0
10564: a0 24 00 00 st Xlo, [ X10 ]
10568: d2 02 3f fc ld [ Xo0 + -4 ], Xo1
1056c: 8e c2 40 00 call Xo1
10570: 01 00 00 00 nop
10574: 9b c3 e0 08 cmp Xo0
10578: d2 02 00 00 ld [ Xo0 ], Xo1
1057c: 80 a2 60 00 cmp Xo0, 0
10580: 12 bf ff f9 bne 0x105b4
10584: 90 02 20 04 add Xlo, 0x4, X0
10588: 11 00 00 00 sethi Xhi(0), X0
1058c: 90 12 20 1c or Xlo, 0x1c, X0 ! 0x1c
10590: d2 05 c0 08 ld [ Xl7 + Xo0 ], Xo1
10594: 80 a2 60 00 cmp Xo0, 0
10598: 02 80 00 06 be 0x1056c
1059c: 13 00 00 00 sethi Xhi(0), X0
105a0: 92 12 60 08 or Xo1, 8, X0 ! 0x8
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:

00020770 <.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
20788: 03 00 00 48 sethi $hi(0x12000), %g1
2078c: 30 bf ff f4 b,a 0x20740
20790: 01 00 00 00 nop
20794: 03 00 00 54 sethi $hi(0x15000), %g1
20798: 30 bf ff f8 b,a 0x20740
2079c: 01 00 00 00 nop
207a0: 03 00 00 60 sethi $hi(0x18000), %g1
207a4: 30 bf ff e7 b,a 0x20740
207a8: 01 00 00 00 nop
207ac: 03 00 00 6c sethi $hi(0x1b000), %g1
207b0: 30 bf ff e4 b,a 0x20740
207b4: 01 00 00 00 nop
207b8: 01 00 00 00 nop
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. Example: *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?

- Each language encourages thinking about a problem in a particular way.
- Each language provides (slightly) different expressiveness & efficiency.

⇒ The language should match the problem.
- Languages give insights into the foundations of computation

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*
Computational Paradigms (Cont.)

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 at the same time

  - functional (task/threads) and data parallelism
  - different granularities: instruction, loop, or task level
  - synchronization: locks, message passing, ...
  - Key notions: *control and data dependencies*
Computational Paradigms (Cont.)

Quantum Computing:

Circuit of qbit gates

- qbits in superposition
- reversible computation
- qbit measurement collapses it to 0 or 1 with a specific probability
- qbit entanglement: “spooky action at a distance” (Einstein)
- beats “classical computing” on some problems

Basis in Quantum theory
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*
Syntax and Semantics of Prog. Languages

Syntax:
Describes what a legal program looks like

Semantics:
Describes what a correct (legal) program means

A formal language is a (possibly infinite) set of sentences (finite sequences of symbols) over a finite alphabet $\Sigma$ of (terminal) symbols: $L \subseteq \Sigma^*$

Examples:

- $L = \{ \text{identifiers of length 2} \}$ with $\Sigma = \{a, b, c\}$
- $L = \{ \text{strings of only 1s or only 0s} \}$
- $L = \{ \text{strings starting with } \$ \text{ and ending with } \# \text{, and any combination of 0s and 1s inbetween} \}$
- $L = \{ \text{all syntactically correct Java programs} \}$

Claim: The larger the language, the harder it is to formally specify the language. In other words, it gets harder for each $i$: $L_1 \subset L_2 \subset L_3 \ldots \subset L_i \subset \ldots$. True or false?
Syntax and Semantics: How does it work?

**Syntactic representation of “values”**

What do the following syntactic expressions have in common?

- XI
- 1011
- B
- $\lambda f. x. (f(f(f(f(f(f(f(f(f(x)))))))))$)
- $\# \# \# \# \# \# \# \#$
- $3 + 20 - (2 \times 6)$
Syntax and Semantics: How does it work?

Syntactic representation of “values”

What do the following syntactic expressions have in common?

XI
1011
B
\( \lambda f x. (f(f(f(f(f(f(f(f(f(f(f))))))))))) \)
$ \|\|\|\|\|\|\|\|\#$
3 + 20 - (2 \times 6)

Answer: They are possible representations of the integer value “11” (written as a decimal number)

What is computation?

Possible answer: A (finite) sequence of syntactic manipulations of value representations ending in a “normal form” which is called the result. Normal forms cannot be manipulated any further.
Syntax without Semantics?

Syntax without semantics is not useful!

Two problems on rewrite systems in the first homework.
Things to Do

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 if you do not have one already

Recitations will start NEXT TUESDAY, JANUARY 24.