198:314 Principles of Programming Languages

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 $\Rightarrow$ 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
$2 \times 30\%$	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.)

- The textbook for this course is *" Programming Language Pragmatics* by Michael L. Scott, 4th Edition, Morgan Kaufmann (Elsevier), 2015.
- 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/ANNOUCEMENTS

⇒ 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.

Course Information (Cont.)

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 **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.

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);
}
```

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
.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
decl %edx
jns .L21
xorl %ecx, %ecx
xorl %eax, %eax
movl $99, %edx
.p2align 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 $.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  %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
```

```
   106a4: 01 00 00 00  nop
   106a8: 81 c7 e0 08  ret
   106ac: 81 e8 00 00  restore
   106b0: 9d e3 bf 90  save  %sp, -112, %sp
   106b4: 81 c7 e0 08  ret
   106b8: 81 e8 00 00  restore
Disassembly of section .init:

000106bc <.init>:
   106bc: 9d e3 bf a0  save  %sp, -96, %sp
   106c0: 7f ff ff b1  call  0x10584
   106c4: 01 00 00 00  nop
   106c8: 7f ff ff e6  call  0x10660
   106cc: 01 00 00 00  nop
   106d0: 81 c7 e0 08  ret
   106d4: 81 e8 00 00  restore
Disassembly of section .fini:

000106d8 <.fini>:
   106d8: 9d e3 bf a0  save  %sp, -96, %sp
   106dc: 7f ff ff 79  call  0x104c0
   106e0: 01 00 00 00  nop
   106e4: 81 c7 e0 08  ret
   106e8: 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
   20788: 03 00 00 48  sethi  %hi(0x12000), %g1
   2078c: 30 bf ff ed  b,a   0x20740
   20790: 01 00 00 00  nop
   20794: 03 00 00 54  sethi  %hi(0x15000), %g1
   20798: 30 bf ff ea  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.

$\Rightarrow$ 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.)

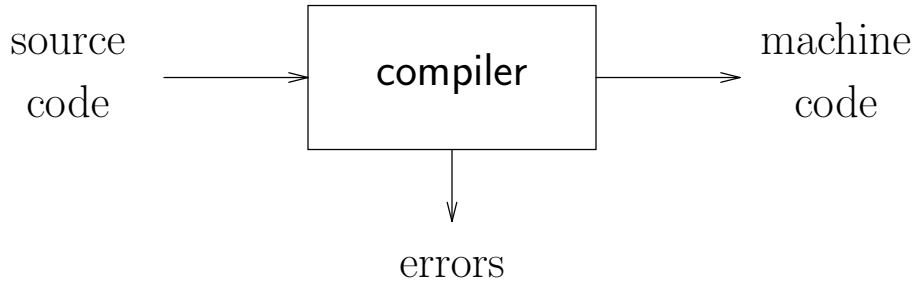
Quantum Computing:

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

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

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 Σ of (terminal) symbols: $L \subseteq \Sigma^*$

Examples:

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

Claim: *The larger the language, the harder it is to formally specify the language. In other words, it get's harder for each i:* $L_1 \subset L_2 \subset L_3 \ldots \subset L_i \subset \ldots$. True or false?

Syntactic representation of "values"

What do the following syntactic expressions have in common?

XI

1011

B

$\lambda fx.(f(f(f(f(f(f(f(f(f(f(fx)))))))))))$

\$ |||||||||| #

$3 + 20 - (2 \times 6)$

Syntactic representation of "values"

What do the following syntactic expressions have in common?

XI

1011

B

$\lambda fx.(f(f(f(f(f(f(f(f(f(f(fx)))))))))))$

\$ |||||||||| #

$3 + 20 - (2 \times 6)$

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 respresentations 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 of you do not have one already

Recitations will start **NEXT TUESDAY, JANUARY 24**.