Course Summary

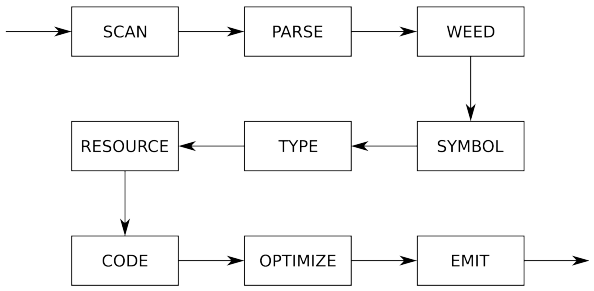
COMP 520: Compiler Design (4 credits)

Alexander Krolik

alexander.krolik@mail.mcgill.ca

MWF 8:30-9:30, TR 1080

http://www.cs.mcgill.ca/~cs520/2019/



Compiley “Mompiler” McCompilerface

Announcements (Monday, April 8/Wednesday, April 10)

Milestones

- Milestone 4 due: Wednesday, April 10th 11:59 PM (2 days grace) on GitHub “milestone4”
- Final submission/report due: Friday, April 12th 11:59 PM (2 days grace) on GitHub “pineapple”
- Group meeting: Week of April 8th (grace until the week of April 15th)
- Peephole due: Friday, April 12th 11:59 PM on myCourses
- Final exam: Thursday, April 18th 2:00 PM

How is everyone doing?

End of Semester!

- **Monday:** Scanner – bytecode generation
- **Wednesday:** Optimization – native code generation + special topics
- Course evaluations! Please let us know what you think of the course!

Why did we learn about Compilers?

Why did we learn about Compilers?

Language design

Look under-the-hood at how code is transformed for execution

Connect theory (automata/CFG) to practice

CS credits, need to graduate

How does learning about compilers change your view of Programming Language Usage/Design/Implementation?

How does learning about compilers change your view of Programming Language Usage/Design/Implementation?

Behaviour is well (usually) specified

Languages have a theoretical base

Type and semantics rules are quirky, but essential to compilers

Funky features are hard to support (Go is weird!)

Be careful! Unintended consequences are easy!

For your next compiler, what advice would you give yourself?

For your next compiler, what advice would you give yourself?

*The AST is **essential***

Modularity: Decouple passes and phases as much as possible

Test, test, test! (automation is key)

Start early!

Don't be afraid of refactoring

*Your work is **never** perfect*

Classes can be fun!

Final Exam (100 points)

1. Scanners and Parsers (22 points)
2. Typechecking (10 points)
3. Bytecode Generation (12 points)
4. Optimization (10 points)
5. Native Code Generation (10 points)
6. Garbage Collection (10 points)
7. Special Topics (10 points)
8. GoLite Project (16 points)

Other info

- Emphasis on application
- Virtual machines (JVM and VirtualRISC) and TinyLang cheatsheets included

Studying Tips

- Review Vince's midterm review
- Review the midterm, if you missed something, review the notes and figure out the right answer
- Practice the 2018 final exam
- Practice real questions like those on the board

Writing tips

- Organize your answers - make is easy to find your answers
- Write neatly, don't squish in your answers to make a lot fit on one page
- Start each question on a new page
- Be precise
- **Manage time wisely**

Scanners+Parsers

Scanners

- Know how to implement a scanner in `flex` or `SableCC`
- Know the limitations and languages that be recognized by regular expressions/DFAs/NFAs

Parsers

- Know what makes a grammar ambiguous, and how to write unambiguous context-free grammars in `bison` or `SableCC`
- Know the limitations and languages that be recognized by context-free grammars

Example

- Give the scanner+parser in either `flex`+`bison` or `SableCC` using no precedence directives

Scanners+Parsers

Write scanner and parser using either `flex+bison` or `SableCC` for the following language.

```
void main(string s) {
   int x, y;                  // comma-separated lists in declarations
   string s;

   x = y = 10;                // an integer constant (no leading zeros)
   s = "Hello" + "World";     // a string constant, concatenation
   write(x);                  // write built-in function, reserved keyword
   {                          // block statements
      y = cube(x);            // function calls
      write(foo(x, y));       // functions can have multiple parameters
   }
}

int cube(int x) {
   return x ** 3;             // exponentiation (right associative)
}

int foo(int x, int y) {
   if (y) {                   // if-else statements
      return y * -1;          // return statements
   } else {
      return x + (3 * y);
   }
}
```

Typechecking

- Know how to express type rules as pseudocode, inference rules, or plain English

Example

- Give the type inference rule for typechecking the + binary expression (assuming we have overloaded string concatenation)
- Give the pseudocode and prose for the following type inference rule for typechecking an assignment

$$\frac{V(x) = \tau \quad V \vdash E : \sigma \quad \tau := \sigma}{V \vdash x = E : \tau}$$

Bytecode Generation

- Know how to generate JVM bytecode (Jasmin syntax): directives, types, instructions, labels, etc.
- Know control flow: loops, if/else, logicals, etc.
- Know how to access fields, and invoke functions/constructors, etc.
- Know how the baby stack works to compute output (some internals of the JVM)

Example

```
public class Computer
{
   protected bool status;

   public bool SetStatus(String s)
   {
      if (wait() == 0)
      {
         return status = s.equals("on");
      }
      return false;
   }

   public int wait();
}
```

What if `status` was a `Boolean` (with appropriate constructor calls)?

Optimization

- Understand why and how generated code is inefficient
- Know how and when to optimize code, while maintaining the original semantics (soundness)
- Peephole optimization (not static analysis)

Example

```
iload_1
iconst_2
iadd
istore_1
```

$\Longrightarrow$ `?`

Design a general peephole pattern for optimizing the above code. What conditions are necessary for this pattern?

Native Code Generation

- Know the 3 different register allocation schemes covered in this class (naïve, fixed, and basic block)
- Understand the register allocations for each scheme and the associated advantages/disadvantages

Example

Given the following code, write the equivalent VirtualRISC code using the naïve and fixed register allocation schemes ($m = n = k = 2$).

```
public static bool isFoo(int p) {
   int iter = p / 2;
   int current = 0;

   while (iter > 0) {
      iter = iter - 1;
   }
   return true;
}
```

Garbage Collection

- Understand workings of garbage collection techniques
- Know the rules for reference counting, mark-and-sweep, and stop-and-copy
  - Scopes, function calls, assignment, etc.
- Know what makes record live and dead

Example

```
public void foo(Object a /* id=1 */) {
   int b = 1337;
   Object c = new Object(2);
   {
      c.SetField_1(a)
      c.SetField_2(new Object(3));

      Object d = new Object(4);
      d.SetField_1(c.GetField_2())
      // (a) GIVE THE REFERENCE COUNTS FOR ALL OBJECTS AT THIS POINT
   }

   // (b) SHOW THE PROGRESSION OF MARK-AND-SWEEP AND STOP-AND-COPY AT THIS POINT
   c = a
   // (c) GIVE THE REFERENCE COUNTS FOR ALL OBJECTS AT THIS POINT
}
```

Special Topics

GPUs

- Understand modern GPU architecture at a *high*-level (threads, concurrency, memory hierarchy, etc.)
- Know the benefits of coalescing memory accesses, and the impact on the number of memory transactions
- Know how to compute a parallel reduction

WebAssembly

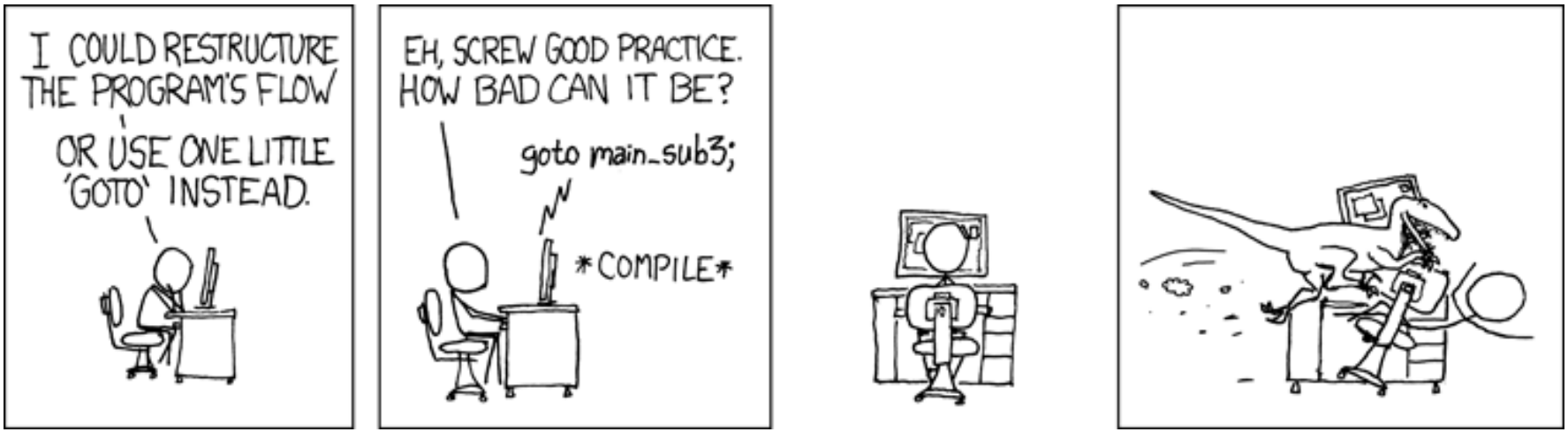
- Understand the underlying implementation (i.e. it is a stack machine) at a *high*-level
- Know how the ISA is validated before execution

Examples

- Look at the slides/notes!

My Thoughts

- Language design is a curious topic, where seemingly innocuous changes suddenly create this...



https://xkcd.com/292/

- Language design is more subtle and complex than the high-level view known to most programmers
- Semantics are fun! (but hard)
- Compilers are fun! (but a lot of work)
- Hopefully this comes in handy one day!
- *One day, reflect on how your view of programming changed*

Thanks...

- To Adrian, who recorded and edited all semester
- To David/Maxence/Mathieu, who worked hard as your TAs
- To Laurie & Clark, for help and support all semester
- To you! This class is a ton of work and you worked hard all semester

“Don’t let the language get in the way of your logic”