CS 6371: Advanced Programming Languages

Dr. Kevin Hamlen

Spring 2020

Today’s Agenda

- Course overview and logistics
- Course philosophy and motivation
  - What is an “advanced” programming language?
  - Type-safe vs. Unsafe languages
  - Functional vs. Imperative programming
- Introduction to OCaml
  - The OCaml interpreter and compiler
  - An OCaml demo

Course Overview

- How to design a new programming language
  - specifying language formal semantics
  - bad language design and the “software crisis”
  - “new” programming paradigms: functional & logic
  - how to formally prove program correctness
- Related courses
  - CS 4337: Organization of Programming Languages
  - CS 5349: Automata Theory
  - CS 6353: Compiler Construction
  - CS 6367: Software Verification & Testing

Course Logistics

- Class Resources:
  - Course homepage: www.utdallas.edu/~hamlen/cs6371sp20.html
  - My homepage: www.utdallas.edu/~hamlen
  - Tentative office hours: 1 hr immediately after each class
  - Email: hamlen AT utdallas DOT edu
- Grading
  - Homework: 25%
  - In-class quizzes: 15%
  - Midterm exam: 25%
  - Final exam: 35%
- Homework
  - 9 assignments: 6 programming + 3 written
  - Homework must be turned in by **1:05pm** on the due date. Programming assignments submitted through eLearning; written assignments submitted in hardcopy at start of class.
  - Late homeworks NOT accepted!
- Attendance of at least 2 of first 3 classes is MANDATORY.

Homework Policy

- Students MAY work together with other current students on homeworks
- **You MAY NOT consult homework solution sets from prior semesters (or collaborate with students who are consulting them).**
- **CITE ALL SOURCES**
  - includes webpages, books, other people, etc.
  - citation is required even if you don't copy the source word-for-word
  - there is nothing wrong with using someone else's ideas as long as you cite it
  - you will not lose any marks or credit as long as you cite
- Violating the above policies is PLAGIARISM (cheating).
- Cheating will typically result in automatic failure of this course and possible expulsion from the CS program.
- It is much better to leave a problem blank than to cheat!
  - Usually ~60% is a B and ~80% is an A.
  - However, cheating earns you an F. It's not worth it!

Quizzes

- in-class on specified homework due dates
- about 15-20 min. each
- approximately 1 quiz per unit, so about 8 total
  - lowest one dropped, so you can miss one without penalty
  - other misses only permitted in accordance with university policy (e.g., illness with doctor’s note, etc.)
- closed-book, closed-notes
- think of them as extensions to the homework
  - length/difficulty similar to one or two homework problems
  - To prepare, be sure you can solve problems like those seen on the most recent homework in about 15-20 minutes each and *without group help!*

Difficulty Level

- Warning: This is a tough course
  - cutting-edge, PhD-level material
  - difficulty ranked very high by past students
- No required text book
  - very few (approachable) texts cover this advanced material
  - no large pools of sample problems exist to my knowledge
  - useful texts:
    - book by Glynn Winskel on reserve in UTD library
    - online text and several online manuals linked from webpage
  - Warning: Some online web resources devoted to this material are INCORRECT (e.g., certain Wikipedia pages). Rely only on *authoritative* sources.
- What you'll get out of taking this course
  - excellent preparation for PhD APL qualifier exam
  - solid understanding of language design & semantics
  - modern issues in declarative vs. imperative languages
  - deep connections between abstract logic and programming

About me…

- PhD & Masters from Cornell University, B.S. in CS & Math from Carnegie Mellon University
- Research: Computer Security, PL, Compilers
- Industry/Government Experience: Microsoft Research; PI for Navy, Air Force, Army, DARPA, NSF, NSA, …
- Personal
  - Christian
  - married, three sons (one 7-year-old, and twin 4-year-olds)
- Programming habits
  - C/C++ (for low-level work)
  - assembly (malware reverse-engineering)
  - C#, Java (toy programs)
  - Prolog (search-based programs)
  - OCaml, F#, Haskell, Gallina/Coq (everything else)

Course Plan

- Running case-study: We will design and implement a new programming language
- Code an interpreter in OCaml
  - OCaml (“Objective Categorical Abstract Meta-Language”) is an open-source variant of ML
  - Microsoft F# is OCaml for .NET (but not fully compatible with OCaml, so don’t use it for homework)
  - Warning: OCaml has a STEEP learning curve!
  - Pre-homework: Install OCaml
    - Go to the course website and follow the instructions entitled “To Prepare for the Course…” by next time

What is an “Advanced” Programming Language?

C/C++: Unsafe Languages

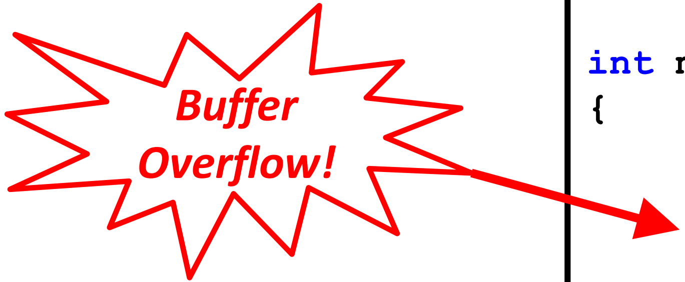
- Find the bug:

```
#include <stdio.h>

int main()
{
	char name[1024];
	printf("Enter your name: ");
	gets(name);
	printf("Your name is: %s\n", name);
	return 0;
}
```

C/C++: Unsafe Languages

- Find the bug:



```
#include <stdio.h>

int main()
{
	char name[1024];
	printf("Enter your name: ");
	gets(name);
	printf("Your name is: %s\n", name);
	return 0;
}
```

- C/C++ lets you write programs that seg fault
- Some language features *cannot* be used safely!
- Most of the software crashes you experience are a direct result of the unsafe design of C/C++

Java: A Type-safe, Imperative Language

- Find two bugs:

```
import java.io.*;
import java.util.*;

class Summation {
  public static void main(String[] args) {
    List list = new LinkedList();

    for (int i=0; i<args.length; ++i)
      list.add(args[i]);

    int sum = 0;
    while (!list.isEmpty())
      sum += ((Integer)list.remove(1)).intValue();

    System.out.println(sum);
  }
}
```

Java: A Type-safe, Imperative Language

- Find two bugs:

OutOfBounds Exception!

```
import java.io.*;
import java.util.*;

class Summation {
  public static void main(String[] args) {
    List list = new LinkedList();

    for (int i=0; i<args.length; ++i)
      list.add(args[i]);

    int sum = 0;
    while (!list.isEmpty())
      sum += ((Integer)list.remove(1)).intValue();

    System.out.println(sum);
  }
}
```

Cast Exception!

Problems with Java

- Every Java cast operation is a potential crash
  - In Java, a “crash” is an uncaught exception instead of a seg fault
- Some typecasting issues can be solved with Generics, but not all (e.g., list emptiness check)
- Problem: Java relies on programmer-supplied typing annotations

Goals of Functional Languages

- In an “Advanced” Programming Language:
  - The compiler should tell you about typing errors in advance (not at runtime!)
  - The language structure should make it difficult to write programs that might crash (no unsafe casts!)
  - 80% of your time should be spent getting the program to compile, and only 20% on debugging
  - should be tractable to create a formal, machine-checkable proof of correctness for mission-critical core routines, or even full production-level apps

In OCaml…

- You almost never need to cast anything
  - The compiler figures out all the types for you
  - If there’s a type-mismatch, the compiler warns you
- OCaml is fast
  - Somewhere between C (fastest) and Java (slow)
  - Very hard to measure precisely. (So-called “language benchmarks” typically call underlying math libraries that aren’t even implemented in the languages being tested!)
- Functions are “first-class”:
  - you can pass them around as values, assign them to variables, …
  - you can build them at runtime (Runtime Code Generation)
- But: The syntax is very weird if you’ve only ever programmed in imperative languages!

OCaml: Getting Started

- OCaml programs are text files (*.ml)
  - Write them using any text editor (e.g., Notepad)
  - Unix: Emacs has syntax highlighting for ML/OCaml
  - Windows: I use Vim (www.vim.org)
- Installing OCaml (see course website)
  - Unix: pre-installed on the department Unix machines
  - Windows: Self-installers for native x86 and for Cygwin
- Two ways to use OCaml:
  - The OCaml compiler: ocamlc (compile *.ml to binary)
  - OCaml in interactive mode (use OCaml like a calculator)
  - Demo…