Today

- Course Overview
- PL Basics and MyPL intro
- Handouts: Survey (Quiz 0) and Exercise 1

Assignments

- HW1 (out, due next Tues)
Course Overview

Course communication

- Website: http://www.cs.gonzaga.edu/bowers/courses/cpsc326
- Code submission site: http://www.cs.gonzaga.edu/dropoff
- Piazza (you’ll receive an invite)

Please be sure to **carefully** read the syllabus

- **Office hours:** TR 1-2:30, W 1:30-2:30, or by appointment
- **Grading:** 35% hw, 5% project, 15% quizzes, 45% 3 exams
- *Programming Languages: Principles and Paradigms,* Gabrielelli and Martini

Exam, Quiz, and Homework basics

- Exams and Quizzes are closed book, closed note, your own work
- You are responsible for understanding how to do the assignments
- You will be tested on material from assignments (written & programming)
- Okay to work on, discuss assignments together (but plagiarism not allowed)
- Must turn in homework individually
Warnings

- Decent amount of reading
- Large amount of programming!
- Course will move quickly

Class Sessions

- Mix of lecture, discussion/Q&A, exercises, and short quizzes
- No electronic devices allowed during lecture, discussion, quizzes
  - Please see me for accommodations
  - Okay to use during exercises
- Quizzes may be given at beginning, middle, or end of lecture
- I’ll try to leave time at start and end of each lecture for questions
  - But better and highly encouraged to ask questions during lecture

Homework

- All assignments must have a cover sheet
- Homework assignments have a programming and a (shorter) written part
- Each graded separately: programming 24 points, written 4 points
Course Topics

This course covers various aspects of programming languages

Focus is on:

- Programming language implementation
- Programming language paradigms (or “families”)
- Programming language design (constructs)
- Hands-on use of different languages

General topics

About 1/2 of course (approximately 7 weeks)

- Language design and implementation (syntax, semantics, interpretation)
- ... while not a compiler class, we'll implement a (fake) language from scratch
- ... learning about design and implementation demystifies languages
- ... can make it easier to learn new languages

About 1/2 of course (approximately 8 weeks)

- Functional programming in Haskell
- Logic programming using ASP (Clingo)
- ... learning these languages will make you a better programmer!
Note on Python ...

Course begins by implementing an interpreter in Python 3

- If you haven’t used Python or are rusty ...
  - Crash into Python – https://stephensugden.com/crash_into_python
  - See course webpage for links

We’ll be starting with Python this week ...

- I’ll go over some aspects needed
- But, you need to know the basic syntax
- E.g., functions, conditions, loops, lists, dictionaries
### Quizzes and Project

**We'll have frequent quizzes**

- No make up quizzes
- Likely more (and some shorter) quizzes than previous semesters

**Why?**

- Give you feel for kind of questions I ask on exams
- Give you feedback on your understanding of material
- Give me feedback on your understanding
- Short quizzes shown to help improve final course grades / retention

**End of semester project on an aspect of programming languages**

- extend the interpreter (new constructs)
- adding new features (e.g., garbage collection)
- rewrite parts using parsing tools (e.g., ANTLR)
- dive deeper into compilation (e.g., LLVM)
- learn a new language, write an application, prepare a tutorial

**More on the project later in the semester ...**

*** Quiz 0 ... survey ***
Exercise ... What is a Programming Language?

One type of definition of a PL (from wikipedia)

“A programming language is a formal language designed to communicate instructions to a machine, particularly a computer.”

“A programming language is a notation for writing programs, which are specifications of a computation or algorithm.”

Another definition: Turing Complete

Turing Machines: Every turing machine consists of:

1. an infinite tape of (blank) memory cells with input string
2. a read/write head
3. a transition function: state, value \(\rightarrow\) new-value, move (L,R), new-state
   - alphabet of symbols (plus blank symbol)
   - set of “states” with a designated start state

Simple Example (assume start one cell to left of first input cell)

- alphabet: \{a, b\}
- states: \(s\) (start), \(s_1, h\) (halt)
- transitions:

<table>
<thead>
<tr>
<th>state</th>
<th>value</th>
<th>new-value</th>
<th>move</th>
<th>new-state</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s)</td>
<td>⊥</td>
<td>⊥</td>
<td>R</td>
<td>(s_1)</td>
</tr>
<tr>
<td>(s_1)</td>
<td>a</td>
<td>a</td>
<td>R</td>
<td>(s_1)</td>
</tr>
<tr>
<td>(s_1)</td>
<td>b</td>
<td>a</td>
<td>R</td>
<td>(s_1)</td>
</tr>
<tr>
<td>(s_1)</td>
<td>⊥</td>
<td>⊥</td>
<td>L</td>
<td>(h)</td>
</tr>
</tbody>
</table>
Exercise example (subtract 1 from a binary number)

• alphabet: \{0, 1\} (binary digits)

• states: \(s\) (start), \(s_1\) (go to end), \(s_2\) (find first 1), \(s_3\) (write 1’s), \(h\) (halt)

• transitions:

<table>
<thead>
<tr>
<th>state</th>
<th>value</th>
<th>new-value</th>
<th>move</th>
<th>new-state</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>R</td>
<td>(s_1)</td>
</tr>
<tr>
<td>(s_1)</td>
<td>0</td>
<td>0</td>
<td>R</td>
<td>(s_1)</td>
</tr>
<tr>
<td>(s_1)</td>
<td>1</td>
<td>1</td>
<td>R</td>
<td>(s_1)</td>
</tr>
<tr>
<td>(s_1)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>L</td>
<td>(s_2)</td>
</tr>
<tr>
<td>(s_2)</td>
<td>0</td>
<td>0</td>
<td>L</td>
<td>(s_2)</td>
</tr>
<tr>
<td>(s_2)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>R</td>
<td>(h)</td>
</tr>
<tr>
<td>(s_2)</td>
<td>1</td>
<td>0</td>
<td>R</td>
<td>(s_3)</td>
</tr>
<tr>
<td>(s_3)</td>
<td>0</td>
<td>1</td>
<td>R</td>
<td>(s_3)</td>
</tr>
<tr>
<td>(s_3)</td>
<td>(\perp)</td>
<td>(\perp)</td>
<td>L</td>
<td>(h)</td>
</tr>
</tbody>
</table>

A language is “Turing Complete” if it can simulate any Turing Machine

• A TM captures a specific algorithm (defines one “machine”)

• Every algorithm can be captured as a TM (Church-Turing thesis)

• A Universal TM is a TM that runs TMs (like computers run programs)

Definition: A PL can express all computations (algorithms)

• Meaning it has the same expressive power as a Universal Turing Machine

• Thus, a language is a PL if it is Turing Complete
Examples of languages that are not Turing Complete:

- Markup languages: HTML, XML, JSON, ...
- Many “domain-specific” languages: SQL, regular expressions

Not necessarily tied to specific constructs

- imperative languages with conditional branching (if-goto, while loops) and arbitrary mem access (# of variables)
- whereas Haskell and LP languages use recursion (no goto, no loops)

“Languages” that are (accidentally) Turing Complete

- Musical Notation (requires human to be the memory/tape)
- Excel spreadsheets w/ formulas (no macros, etc.)
- Apache rewrite rules
- Magic The Gathering card game (human selects moves)
- PowerPoint animations (requires human to follow links)
Brief intro to “MyPL”

Basics

- Made up language for exploring language implementation ideas
- Mixes explicit and implicit typing (for type checking)
- Has most basic constructs you would find in a PL

MyPL Primitive Types and Variables

- supports integers, floats, Booleans, and strings

- variable declarations:
  
  ```plaintext
  var x = 0; # type of x is implicit (int)
  var x: int = 0; # type is explicit (int)
  var str_1 = "a"; # implicit string
  var flag: bool = false; # bool
  var pi5 = 3.14159; # float
  ```

- variable assignment:
  
  ```plaintext
  set x = 42;
  set str_1 = "Hello World!";
  ```

MyPL Expressions

- supports basic mathematical expressions (conservatively typed)

  ```plaintext
  set x = (80 / 2) + 2;
  set str_1 = str_1 + "bc";
  var remainder = 10 % 4; # remainder has type int
  ```
MyPL Loops and If Statements

- while loops along with if, if-else, if-elif, and if-elif-else

```plaintext
while x > 0 do
    print("x is greater than 1");
    set x = x - 1;
end

if x > 0 then
    print("x is large");
    x = x - 1;
elif x == 0 then
    print("x is zero");
    x = 1;
else
    print("x is small");
    x = x + 1;
end
```
MyPL Functions

• all params must have a type, return type inferred

```plaintext
fun add_one(x: int)
    set x = x + 1;
    return x;
end

fun concat(s1: string, s2: string)
    return s1 + s2;
end

var x1 = add_one(0);
set str1 = concat("a", "bc");
```

• functions can be overloaded (same name, different args)

• built-ins: length, get, print, reads/readi/readf, tostr, tofloat

```plaintext
var n = length("abc");    # returns 3
var c = get(0, "abc");    # returns "a" ("" if bad index)

print("Hello World!\n");  # outputs given string

print("Enter your name: ");
var name = reads();        # waits for user input (string)
print("Enter your age: ");
var age = readi();         # waits for user input (int)

var s1 = tostr(42);       # converts int to string
var s2 = tostr(3.14);     # converts float to string
var f1 = tofloat(42);     # converts int to float
```
MyPL Structs

- list of variable declarations

```plaintext
struct Person
  var name = ""; # implicit type and default value
  var age = 0;
end

struct Node
  var val = 0;
  var next: Node = nil; # nil is a special value
end

var joe_king = new Person;
joe_king.name = "Joseph King";
joe_king.age = 42;

var n2 = new Node;
n2.val = 42;
var n1 = new Node;
n1.val = 41;
n1.next = n2;
```

MyPL Programs

- we only consider single-file programs
- normal scoping rules (more later)
- structs and functions have to be defined outside of other functions

The devil is in the details ... (which we'll get into later)