Course Introduction

Filling in today for Ranjit Jhala

April 01

A Programming Language

- Two variables
  - x, y
- Three operations
  - x++
  - x--
  - (x=0)? L1:L2;

L1: x++;
    y--;
    (y=0)? L2:L1
L2: ...

Fact: This is “equivalent to” to every PL!

Good luck writing quicksort
  … or Windows, Google, Spotify!

So Why Study PL?

“A different language is a different vision of life”
- Federico Fellini

Programming Language
Shapes
Programming Thought
So Why Study PL?

Programming Language Affects How:
• Ideas are expressed
• Computation is expressed

Course Goals

“Free your mind”
- Morpheus

New ways to:
- describe
- organize
- think about computation

Learn New Languages / Constructs

Write Programs That Are:
• Readable
• Correct
• Extendable
• Modifiable
• Reusable
Learn How to Learn New PLs

No Java, C# 20 years ago
AJAX? Python? Ruby? Erlang? F#? ...

Learn the anatomy of a PL
- Fundamental building blocks
- Different guises in different PLs

Re-learn the PLs you already know!

Learn How to Design New PLs

... “Who, me?”

Buried in every extensible system is a PL
- Emacs : Lisp
- Word, Powerpoint : Macros, VBScript
- Unreal : UnrealScript (Game Scripting)
- Facebook : FBML, FBJS
- SQL, Renderman, LaTeX, XML ...
Choose Right Language

Learn How to Choose Right PL

“... But isn’t that decided by

• libraries,
• standards,
• and my boss?”

Yes!

My goal: educate tomorrow’s tech leaders & bosses, so you’ll make informed choices

Speaking of Right and Wrong...

Imperative Programming
x = x+1

Imperative = Mutation

WTF?

Bad!

x = x+1

Imperative = Mutation
John Carmack
Creator of FPS: Doom, Quake,…

Don’t Take My Word For It

Tim Sweeney (Epic, Creator of UNREAL)
“In a concurrent world, imperative is the wrong default”

Functional Programming
No Assignment.
No Mutation.
No Loops.
OMG! Who uses FP?!?

So, Who Uses FP?

MapReduce

Linq, F#

Erlang

So, Who Uses FP?
So, Who Uses FP?

Scala

So, Who Uses FP?

OCaml

CSE 130!

So, Who Uses FP?

Course Staff

Instructor: Ranjit Jhala

TAs: Eric Seidel

Tutors: Patrick Torbett, Derek Huynh, Lucas Cycon

Office Hours TBD, Check Website ...

Daniel Ricketts
Course Website

cseweb.ucsd.edu/classes/sp14/cse130-a/

- Nothing printed, everything on the web
- Feel free to bring a laptop to class

Discussion Board

- Piazza will be used for all questions, announcements, clarifications, etc.
- Check often!
- If you haven’t received a request to join, email jhala@cs.ucsd.edu

Discussion Sections

Wednesdays
5:00pm to 5:50pm
6:00pm to 6:50pm
SOLIS 104 (same as lecture)

Starts Tomorrow!

Clickers + Peer Instruction (ish)

Quick Poll:

How many people do not already have an i>clicker?
Clickers + Peer Instruction (ish)

- Make class interactive
  - Help YOU and ME understand what’s tricky

- Everyone must bring an i>clicker to class
  - by Tues 4/8 (we’ll “practice” until then)
  - available at bookstore

- Seating in assigned groups
  - Check course webpage

In-Class Clicker Exercises

1. Solo Vote: Think for yourself, then vote

2. Group Discussion: Groups of ~3 students
   - Practice analyzing, talking about tricky notions
   - (Try to) reach consensus
   - If you have questions, raise your hand!

3. Group Vote: Everyone in group votes

4. Class Discussion:
   - What did you find easy/hard?
   - Questions from here show up in exams

In-Class Clicker Exercises

- Participation counts for 5% of your grade
- Respond to 75% of the questions throughout the quarter
- So, don’t fret if you miss a class or two
- Register your clicker! (check webpage)

Grading and Exams

- No Official “Before-Class” Homework
- In-Class Exercises: 5%
- Programming Assignments (6-8): 30%
- Midterm: 30%
- Final: 35%
Programming Assignments

Schedule up on webpage

Usually due on Mondays or Fridays at 5:00 PM

You may use up to four late days total
- Each late day is a “single” or “whole unit”
- 5 mins late = 1 late day
- Plan ahead, no other extensions

No Textbook

- Online lecture notes
- Resources posted on webpage
- Pay attention to lecture and section!
- Do assignments yourself!

Plan for Next 10 Weeks

1. FP, OCaml, ~5 weeks
2. OO, Scala, ~4 weeks
3. Logic, Prolog, ~1 week

Programming Assignments

Unfamiliar languages
+ Unfamiliar environments

Start Early!
No Compile, No Score

Programming Assignments

Forget Java, C, C++, ...
... other 20th century PLs
Don’t complain
... that OCaml is hard
... that OCaml is @!%@#!

Immerse yourself in new languages!

Free your mind.

Word From Our Sponsors

- Programming Assignments done ALONE
- We use plagiarism detection software
  - Have code from all previous classes
  - MOSS is fantastic, plagiarize at your own risk
• Programming Assignments done ALONE
• We use plagiarism detection software
  • Have code from all previous classes
  • MOSS is fantastic, plagiarize at your own risk
• Zero Tolerance
  • offenders punished ruthlessly
• Please see academic integrity statement
• Click Fraud is also not allowed!

To Ask Me Questions!

Say Hello to OCaml

```c
void sort(int arr[], int beg, int end)
{
    if (end > beg + 1)
    {
        int piv = arr[beg];
        int l = beg + 1;
        int r = end;
        while (l != r-1)
        {
            if(arr[l] <= piv)
            l++;
            else
            swap(&arr[l], &arr[r--]);
        }
        if(arr[l]<=piv && arr[r]<=piv)
        l=r+1;
        else if(arr[l]<=piv && arr[r]>piv)
        {l++; r--;}
        else if (arr[l]>piv && arr[r]<=piv)
        swap(&arr[l++], &arr[r--]);
        else
        r=l-1;
        swap(&arr[r--], &arr[beg]);
        sort(arr, beg, r);
        sort(arr, l, end);
    }
}
```

Quicksort in C

```ocaml
let rec sort xs =
match xs with
[ ] -> []
| h::t ->
  let (l,r) = List.partition ((<=) h) t in
  (sort l) @ h::(sort r)
```

Quicksort in OCaml

Word From Our Sponsors
Say Hello to OCaml

Readability matters...

```ocaml
let rec sort xs =
  match xs with
    [] -> []
  | h::t ->
      let (l,r) = List.partition ((<=) h) t in
      (sort l) @ h::(sort r)
```

Plan for Next ~5 weeks

1) Fast Forward
   • Rapid introduction to what’s in OCaml

2) Rewind

3) Slow Motion
   • Go over the pieces individually

CSE 130 [Spring 2014]
Programming Languages

Introduction to OCaml

Ravi Chugh
UCSD CSE
Computer Science and Engineering
ML: History and Dialects

“Meta Language”
Designed by Robin Milner
To manipulate theorems & proofs

Several dialects:
• Standard ML (SML)
  • Original syntax
• Objective Caml: (OCaml)
  • “The PL for the discerning hacker”
  • State-of-the-art, extensive library, tool, user support
• F# (ML + .NET) released in Visual Studio

Interacting with ML

“Read-Eval-Print” Loop (REPL)

Repeat:
1. System reads expression \( e \)
2. System evaluates \( e \) to get value \( v \)
3. System prints value \( v \) and type \( t \)

What are these expressions, values and types?

ML’s Holy Trinity

Expression \( \rightarrow \) Value

• Everything is an expression
• Everything evaluates to a value
• Everything has a type

OCaml REPL

• Demo: ocaml-top on ieng6
• Extended demo in Section tomorrow
• We will collect data from your submissions to learn about usability of error messages, etc. in practice
  ○ More details to follow…
Complex expressions using “operators”: (why the quotes ?)

• +, -, *
• div, mod

Base Type: Integers

2
2+2
2 * (9+10)
2 * (9+10) -12
2 4 38 26
int

Complex expressions using “operators”: (why the quotes ?)

• Concatenation ^

"ab" "ab" "abxy"
"ab" ^ "xy"
string

Base Type: Strings

true
false
1 < 2
"aa" = "pq"
("aa" = "pq") && (1<2)
("aa" = "aa") && (1<2)
true
false
false
true

Base Type: Booleans

(2+3) || ("a" = "b")
"pq" ^ 9
(2 + "a")

Type Errors

Untypable expression is rejected

• No casting, No coercing
• Fancy algorithm to catch errors
• ML’s single most powerful feature (why ?)
ML’s Holy Trinity

Expressions (Syntax) \[\rightarrow\] Values (Semantics)

Compile-time  
(“Static”) 

Types

Run-time  
(“Dynamic”) 

1. Enter an expression \(e\)
2. ML infers a type \(t\) or emits an error
3. ML evaluates expression \(e\) down to a value \(v\)
4. Value \(v\) is guaranteed to have type \(t\)

Complex Type: Tuples (Products)

- Pairs, Triples, Quadruples, ...
- Nesting:
  - Everything is an expression
  - Nest tuples in tuples

So Far, A Fancy Calculator...

... What do we need next?

Expressions: (2+2, 7>8); (4,false)

Values: (9-3,“ab”^“cd”, (2+2, 7>8)); (6,“abcd”,(4,false))

Types: int * bool
Complex Type: Lists

- Unbounded size
- Can have lists of anything (e.g. lists of lists)
- But ...

All elements must have same type

List operator “Cons” ::

Can only “cons” element to a list of same type

Cons operator

Nil operator

[(1, "a" "b") 3+4; c] ;
[1;2;3;4;5;6] ;
[1;2;3] ;
["a","b","c"] ;
[1;2;3] ;
[1;2;3] ;

Cons operator

Nil operator

Can only “cons” element to a list of same type

Cons operator
Complex Type: Lists

List operator “Append” @

1 @ [2;3];

[]@[1];

[1]

Can only append two lists...

... of the same type

[1] @ [“a”;“b”];

List operator “Head” List.hd

ML types can’t catch some errors though...

hd []

Exception: Failure “hd”.

(ML does infer a type...)

Complex Type: Lists

List operator “Tail” List.tl

The tail of empty list is a run-time error...

tl []

Exception: Failure “hd”.

Expressiveness of type systems is an active area of research!

Lists: Deconstruct (or Destruct)

Head

\( e : T \text{ list} \)
\( \text{hd} e : T \)
\( e \Rightarrow v_1:v_2 \)
\( \text{hd} e \Rightarrow v_1 \)

Tail

\( e : T \text{ list} \)
\( \text{tl} e : T \text{ list} \)
\( e \Rightarrow v_1:v_2 \)
\( \text{tl} e \Rightarrow v_2 \)

(hd [[];[1;2;3]]) = (hd [[];[“a”]])

\( e_1 : T \text{ list} \)
\( e_2 : T \text{ list} \)
\( e_1 = e_2 : \text{ bool} \)
Recap: Tuples vs. Lists

What’s the difference?

- **Tuples:**
  - Different types, but **fixed** number:
    - (3, “abcd”)  
    - (int * string)
    - pair = 2 elts
    - (3, “abcd”, (3.5, 4.2))  
    - (int * string * (float * float))
    - triple = 3 elts

- **Lists:**
  - **Same** type, **unbounded** number:
    - [3;4;5;6;7]  
    - int list

- **Syntax:**
  - Tuples = **comma**  
    - Lists = **semicolon**