CS 421 – Programming Languages and Compilers

Welcome!

Today's class:

- Staff
- Why 421?
- Class structure and policies
- Intro to OCaml
- Assignment of hwk 1 (due Friday)

Staff

Professor: Sam Kamin, 4237 SC, 3-8069
Office hrs: TBD

TAs: Dongyun Jin (TBD)
Baris Aktemur (TBD)
William Mansky (TBD)

Web page:
http://www.cs.uiuc.edu/class/cs421/
Newsgroup: news.cs.uiuc.edu – class.cs421

Why 421?

Why learn about compilers?

- Complete picture of how programs go from keyboard to execution
- Understand translation from high-level language to machine language
- Learn to build compilers and other programs that process structured input
- Learn interesting algorithms

Why 421?

Why learn about languages?

- Increase ability to learn new languages
- Learn correct terminology for describing languages
- Become better programmers by seeing different perspectives on programming

Class structure

- Lectures Wednesday and Friday. Notes posted after class. *Strongly advise taking notes in class.*
- Weekly assignments
- Programming in OCaml
- Grades
- Cheating
- Lateness

Class structure

Topics:

- First half: Compilers
  - First 2 ½ classes: OCaml
- Second half: Languages

Full details on web page

OCaml

- Functional programming language
  - One of the two leading language paradigms (the other is object-oriented)
  - Defined mainly by *no assignment statements*
    - **Heavy use of *dynamically-allocated data structures* and *recursion***
- Everything we will do in the first half of the class could also be done in Java, but:
  - OCaml notationally much more concise
  - Using OCaml now will prepare you for more advanced uses of OCaml in second half

OCaml

- Interactive system

```
> ocaml
# 3+4;;
  7 : int
```

- Load files: #use "filename";;

```
# #use "...";;
```

OCaml

- **Define variables and functions**

```
# let x = 3;;
# let f x = x+1;;
# let f y = y+x;;
# let x = 4;;
# let g y = y+x;;
```

f 10 ⇒ 13

Java:
```
int f(int x){
  return x+1;
}
```

Java:
```
int f(int y){
  return y+x;
}
```

- **Functions of multiple arguments:**

```
let f x y = x+y;;
f: int -> int -> int
```

Java:
```
int f(int x, int y){
  return x+y;
}
```

OCaml

- Arithmetic and comparison operators: usual.
- Boolean operators: &&, ||, not
- Conditional expressions

= , not ==

```
let f x = if x>0 then x+10
          else x*3;;

f 3;;
13 : int
```

Java: int f(int x) {
  return x>0 ? x+10
             : x*3;
}

OCaml

- Use of parentheses

Java: Precedence, Function appli., Cast operator

Only for precedence

(f x) + 3 f (x + 3) (f x) + 3

- Strings: "....", ^ for concatenation. (String module contains length, get. etc.)

f (x) + 3

silly

```
# open String ;;
# length "..." ;;
```

OCaml

- Characters: 'c' (int_of_char, char_of_int)

'c' + 1 int_of_char 'c' + 1

- Printing to console: print_int, print_string

OCaml

- Sequencing: use ;.

```
let f x = x+1 ;;
let g = f ;;
let f x = print_int x; x+1 ;;
let g y = y+1; y+2 ;;
```

- Comments: (* ... *)

Like "old-style" C comments: /* */

except they nest: (* ... (* ... *) ... *)

/* ... /* ... */ ... */

OCaml

- Recursive function definitions

let rec f x = ... f (x-1) ... -

Homework 1 – due Friday

- Define a few simple functions.