Welcome - CSC 301

CSC 301- Foundations of Programming Languages

- Instructor: Dr. Lutz Hamel
- Email: lutzhamel@uri.edu
- Office Hours: WF 11-noon, remotely
- TA: Siyi Li

(for more details see course website)

Why Study Programming Languages?

- Amazing variety
  - ~2300 different programming languages discussed on online forums*.
- "Strange" controversies
  - Should a programming language have a 'goto' statement?
  - Should an OO language support inheritance?
  - Terminology: argument vs. actual parameter.
- Many connections
  - Programming languages touch upon virtually all areas of computer science: from the mathematical theory of **formal languages** and **automata** to the implementation of **operating systems**.
- Intriguing evolution
  - Programming languages change!
    - New ideas and experiences trigger new languages.
    - New languages trigger new ideas, etc.

*Source: Webber, Modern Programming Languages: A Practical Introduction.

Programming Language Classes

There are many different programming language classes, but three classes or paradigms stand out:

- Imperative Languages
- Functional Languages
- Logic/Rule Based Languages

What Happened to OOP?

- Object-orientation is really a property of the type system of a language.
- OO features have traditionally been added to imperative languages (C++, Java, Python)
- Object-oriented features have also been added to:
  - Functional programming languages like Lisp (CLOS)
  - Logic languages like Prolog (Logtalk)
- Here we look at object-orientation in terms of another imperative language - Rust

Meet Our Languages

- Rust – Imperative with OOP features
- Haskell – Functional
- Prolog – Logic

Example Computation

- Recursive definition of the factorial operator

$$x! = \begin{cases} 1 \text{ if } x = 1, \\ x(x-1)! \text{ otherwise.} \end{cases}$$

for all $x > 0$.

Imperative Languages

- Hallmarks: assignment and iteration
- Examples: C, FORTRAN, Rust
- Example Program: factorial program in Rust

```
fn fact(mut n: i32) -> i32 {
    let mut val = 1;
    while n > 1 {
        val = val*n;
        n = n - 1;
    }
    return val;
}
```

Imperative Languages

Observations:

- The program text determines the order of execution of the statements.
- We have the notion of a '*current value*' of a variable – accessible state of variable.

This is not always true in other languages.

Rust

```rust
// factorial program that prints out
// the factorial of x

fn main() {
    let x = 3;
    println!("The factorial of {} is {}", x, fact(x));
}

fn fact(mut n: i32) -> i32 {
    let mut val = 1;
    while n > 1 {
        val = val*n;
        n = n - 1;
    }
    return val;
}
```

Functional Languages

- Hallmarks: recursion and *single valued variables*.
- Examples: ML, Lisp, Haskell
- Example Program: factorial program in Haskell

```
fact 1 = 1
fact n = n * fact (n-1)
```

recursion

Functional Languages

Observations:

- There are **no** explicit assignments.
- The name stems from the fact that programs consist of *recursive* definitions of **functions**.

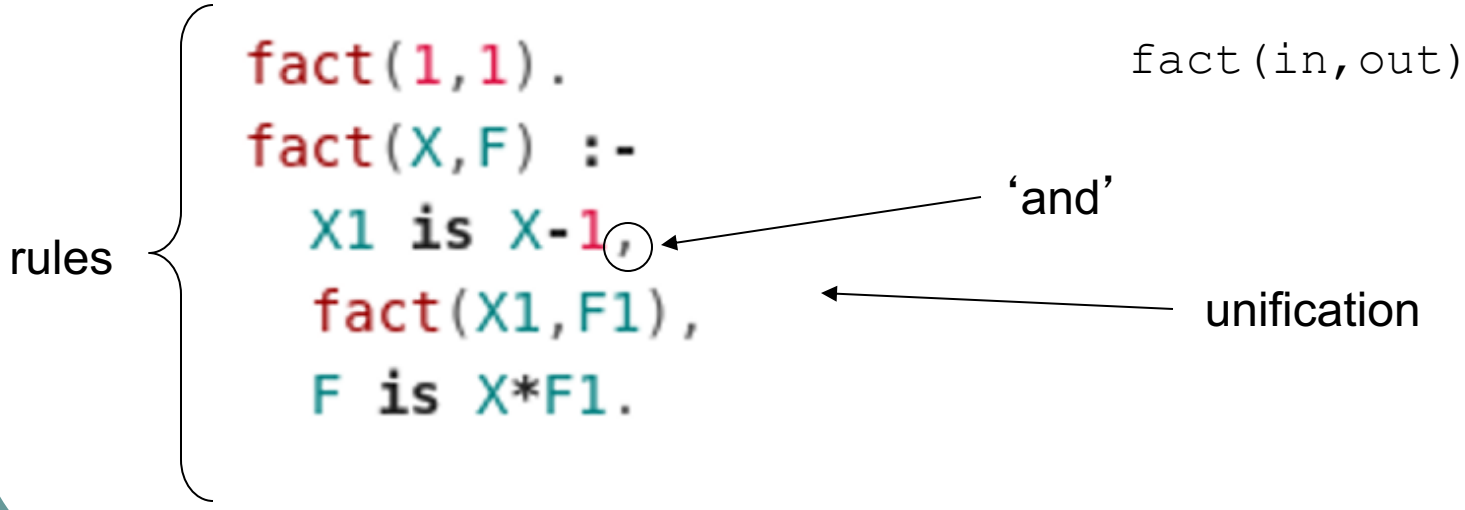
Haskell

```haskell
-- factorial program
compute = do
  let x = 3
  print (fact x)

fact 1 = 1
fact n = n * fact (n-1)
```

Logic Programming Languages

- Hallmarks: programs consist of **rules** that specify the problem solution.
- Examples: Prolog, Maude, Isabelle
- Example Program: factorial program written in Prolog



Logic Programming Languages

Observations:

- Rules do *not* appear in the order of execution in the program text.
- No specific order of execution is given – rules ‘fire’ when necessary.

Prolog

```
% factorial program

fact(1,1).
fact(X,F) :-
  X1 is X-1,
  fact(X1,F1),
  F is X*F1.

compute :-
  X is 3,
  fact(X,F),
  writeln(F).
```

Object-Oriented Languages

- Hallmarks: bundle data with the allowed operations ☞ Objects
- Rust takes an interesting approach here – structures with functions.

```
// Simple OO program

struct Rect { xdim: i32, ydim: i32}

impl Rect {
  fn area(&self) -> i32 { return self.xdim*self.ydim}
}

fn main() {
    let rect = Rect{xdim:3, ydim:4};
    println!("The area is {}", rect.area())
}
```

Programming Language Classes

General Observations:

- Programming languages guide programmers towards a particular programming style:
  - Imperative → iteration/assignment
  - Functional → mathematical functions
  - OO → objects
  - Logic → rules
- Programming itself guides the developer towards new language ideas:
  - Recursion was introduced by John McCarthy in the 1950's with the programming language Lisp to solve problems in AI.
  - Classes and objects were developed by Nygaard and Dahl in the 1960's and 70's for the language Simula in order to solve problem in simulations.

Take Away

- There exist many programming languages today (> 2000)
- In order to understand the similarities and differences $\Rightarrow$ sort into *classes*
  - Imperative
    - assignment and iteration
  - Functional
    - Recursion, single valued variables
  - Logic/rule based
    - programs consist of rules
- Object-oriented
  - bundle data with the allowed operations

Reading & Assignments

- Reading: Intro from “A Gentle Introduction to Rust” from here on known as GIR
  - https://stevedonovan.github.io/rust-gentle-intro/readme.html
- Assignment #0: Download & Read Syllabus – upload a copy of it into BS