TDDE55 Data and Program Structure Introduction

Cyrille Berger

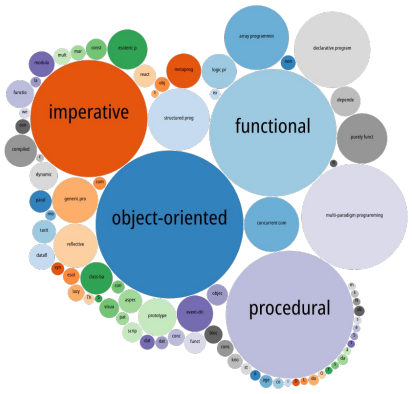
LINKÖPING UNIVERSITY

Course Introduction

Lecture content

- Course Introduction
- Course Practicalities

Course goals

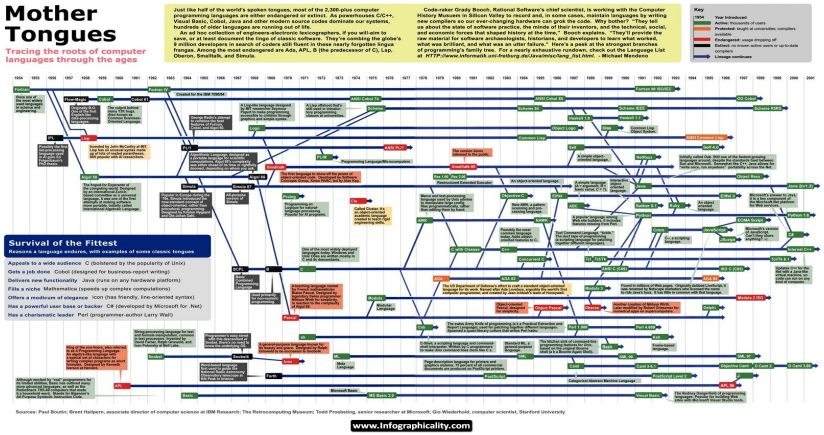


- Describe aspects of **evaluation** and **execution** in different language models
- Explain and demonstrate how design choices affect the **expressiveness** and efficacy of a programming language
- Analyze and value programming languages based on their **evaluation** and **compilation** strategies
- Implement programming languages in the form of an **interpreter** and a **compiler**

Why should you care about this course?

- Programming languages are the key tools that define how we think about computations, and what we are able to
- It will help you understand why programming language works a certain way and what are the limits
- What you learn during this course will give you the power to learn, choose, and craft your tools effectively.
- In this course we will look at programming with a scientific eye.

Evolution of programming languages



Creative extension principle

- Why new language/new extension?


- Existing language have limitations in expressiveness
- With increase in complexity of the features provided, the source code complexity increase, this can be solve with **new programming concept**

Example of creative extension princip

- Without exception

```
function f1(a, b, c)
{
  cond(b==0, return(false));
  ...
  return(true);
}

function f2(a, b, c)
{
  define d = f1(a, b, c);
  cond(d, return(false));
  ...
  return(true);
}

function f3(a, b, c)
{
  define d = f2(a, b, c);
  cond(d, return(false));
  ...
  return(true);
}

cond(f3(a,b, c), print("division by 0"));
```

- With exception

```
function f1(a, b, c)
{
  cond(b==0, raise("Division by 0"));
  ...
  return();
}

function f2(a, b, c)
{
  define d = f1(a, b, c);
  ...
  return();
}

function f3(a, b, c)
{
  define d = f2(a, b, c);
  ...
  return(true);
}

catch(f3(a,b, c), function(except) {
print(except) });
```

The story of null

- Zero was inventend in India by Brahmagupta in the 7th Century
  - A revolution in mathematics, from LXXXIX to 89…
  - A placeholder for nothing
  - Critical development in number theories
  - …
- *Null* was invented in 1964 by Tony Hoare for ALGOL-60
  - It is also a placeholder for nothing

The Billion Dollar Mistake

- *Null* is an example of *creative extension* gone wrong
  - It was added to by-pass safety checks

  Tony Hoare: *I call it my billion-dollar mistake…At that time, I was designing the first comprehensive type system for references in an object-oriented language. My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.*

What is wrong with null? (1/4)

- NULL subverts types
  - In Java x.toUpperCase() can be called as long as x is a string… and is not null
- NULL is sloppy
  - `if(x == null || x.equals(""))`
  - `if(string.IsNullOrEmpty(x))`

What is wrong with null? (2/4)

- NULL is a special-case
  - ```
    char c = 'A';
    char *myChar = &c; // works
    std::cout << *myChar << std::endl; // works
    char *myChar2 = 123; // compile error
    char *myChar3 = 0; // valid
    std::cout << *myChar3 << std::endl; // runtime error
    ```
- In C strings are null terminated…
    ```

What is wrong with null? (3/4

- NULL makes poor APIs
  - Key-value store

```
class Store
    ##
    # associate key with value
    #
    def set(key, value)
        ...
    end

    ##
    # get value associated with
key, or return nil if there is
no such key
    #
    def get(key)
        ...
    end
end
```

```
store = Store.new()
store.set('Bob',
'801-555-5555')
store.get('Bob') # returns
'801-555-5555', which is
Bob's number
store.get('Alice') # returns
nil, since it does not have
Alice
store = Store.new()
store.set('Ted', nil) # Ted
has no phone number
store.get('Ted') # returns
nil, since Ted does not have
a phone number
```

- Double nulls

What is wrong with null? (4/4

- NULL exacerbates poor language decisions

```
int x = null; //
compile error
Integer i = null;
int x = i; //
runtime error
```

- NULL is difficult to debug

```
#include <iostream>
struct Foo {
    int x;
    void bar() {
        std::cout << "La la la"
<< std::endl;
    }
    void baz() {
        std::cout << x <<
std::endl;
    }
};
int main() {
    Foo *foo = NULL;
    foo->bar(); // okay
    foo->baz(); // crash
}
```


How to solve the NULL problem

- With some *creative extension design*...
- Compile check (in Crystal):

```
my_string = None
if rand(2) > 0
  my_string = "hello
world"
end
puts
my_string.upcase
```

- Optional<T>

  Contains either a value, or information if the value is set

```
cache = Store.new()
cache.set('Bob', Some('801-555-5555'))
cache.set('Tom', None())

bob_phone = cache.get('Bob')
bob_phone.is_some # true, Bob is in cache
bob_phone.get.is_some # true, Bob has a
phone number
bob_phone.get.get # '801-555-5555'

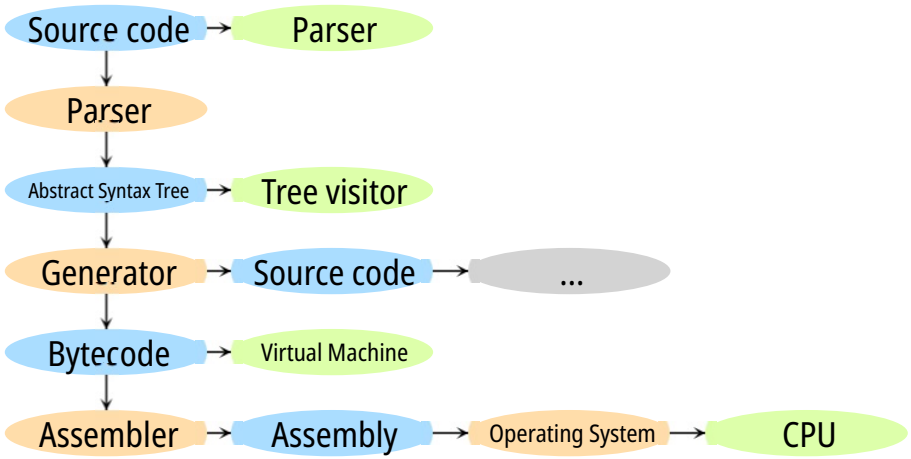
alice_phone = cache.get('Alice')
alice_phone.is_some # false, Alice is not
in cache

tom_phone = cache.get('Tom')
tom_phone.is_some # true, Tom is in cache
tom_phone.get.is_some #false, Tom does
not have a phone number
```


The moral of the story

- *creative extension design* brings good and bad changes
- New concepts need to be tested
- Design choices in programming languages have far reaching conscequences

How is a program interpreted?



Facebook's work on the PHP intepreter

- Facebook started with PHP in 2004
- Back at the time, PHP was the gold standard for website programming and prototyping

- But this is causing problems and for practical reasons they cannot change programming language

What did Facebook do?

- The standard PHP interpreter is using a virtual machine (Zend)
- They developed a tool to convert PHP to C++
- Then they developed a new interpreter that do Just-In-Time (JIT) compilation, called HHVM
- They introduced *Hack*, a variant of PHP with a typing system

And other examples...

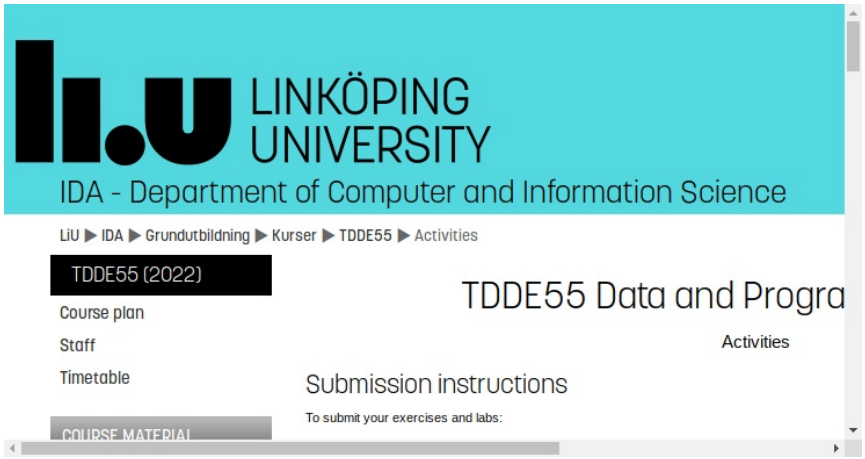
- Google with Java, Dalvik, ART...
- Python with CPython vs pypy...
- Qt's JavaScript, switching from AST Interpretation to JIT and to a mix of JIT and AST Interpretation
- ...

Course Practicalities

Inverted Classroom

- Main idea is that traditional homework is done in the class, and traditional class activities are done at home
- Lectures are all as the video
- You are free to conduct exercises during seminar/labs or at home
- You can ask questions during seminar/labs and I will setup a forum on lisam

Teaching Activities



Division of time

- ~15h lectures (in 17 sessions)
- ~48h labs/tutorials (in 24 sessions)
- ~107h homework
- Labs: maximum 2 per group

Change from last year

- Change of course code
- First year with pre-recorded lectures, an average of 10-15 students watching them, up from 0-2 for live lectures
- Update to Exercise 1
- One lecture will get major changes
- Some labs will get more detailed instructions/examples

And now what?

- Watch lectures, read the book, do the exercises
- Come/attend to the labs to ask questions