CS1101S - Programming Methodology I

Studio 4

Theodore Leebrant

Tutorial Group 8D

Admin Stuff

Attendance Taking

Make sure you have taken your temperature.

We will take photo when everyone is present.

Mastery Checks

Mastery Check 1 is open.
Please telegram me to set a schedule.
Details are in the slides for Studio 2.

Events

- Contest #1 - Closes 1st September
- Reading Assessment - 4 September, 1030 (1000) - 1114

About the Reading Assessment

- LumiNUS Quiz
- One sheet of A4 paper
- Prepare zoom, recording app, matric card, pencil and eraser / pen
- Practice questions available in LumiNUS

Question: do you want to go through the questions in past paper RA1?

Recap

Anonymous functions / Lambda expression

```
const f = param => param + 1;

function f(param) {
  return param + 1;
}
f(1);
```

Function inside functions

```
function factorial(n) {
  function fact_helper(n, res) {
    return n === 1
      ? res
      : fact_helper(n - 1, n * res);
  }

  return fact_helper(n, 1);
}
```

Higher order functions

Having functions as arguments and / or returned value.
Non-programming example: integration / differentiation.

Higher order functions

Say I want the smaller of two things:

```
const min = (a, b) => a < b ? a : b
```

But what if I want to compare two times? (e.g. in hh:mm format)

```
const min = (f, a, b) => f(a) < f(b) ? a : b
```

Higher order functions

Partial application of function:

```
const sum = a => b => a + b;
const add_3 = sum(3);
add_3(100);
```

Scoping

- We give names to things
- We may give many things the same name
- Which names refer to what? (we need context)

Scoping

- A name occurence refers to the closest surrounding declaration
- Scope is the context where we find out names
- Most common context: blocks `{ ... }`
- To find what a name refers to, look at the current scope, and then outwards. Take the first one you come across.
- Names in an outer scope can be hidden by definitions in an inner scope.

Scoping

What makes a scope?

Scoping

```
const hello = "world";
function n(hello) {
    const g = hello => display;
    g(hello)(hello);
    return g(hello);
}
n("hello")(hello);
```

Scoping

```
const n = 1;
{
    const n = 2;
    {
        const n = 3;
        {
            display(n);
        }
    }
}
```

Scoping

```
(f => x => f => x => f(x))
    (x => x + 1)(3)(x => x)(x => 2 * x + 3)

(a => a => a => a)(a => a)(a => a => a)(a => a => a)

(x => y => z => y(z)) (x => y => x(y)) (y => z => z) (1);
```

Example of HOF: series function

How to model a series $S(n) = \sum_{i=0}^{n} a_i x^n = a_0 x^0 + a_1 x^1 + \ldots$

```
function series_generator(limit, coefficient) {
    // body here
}
function factorial(n) {
  return n * factorial(n - 1);
}
```

Given that $e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$

```
const exp_coeff = n => 1 / factorial(n);
const exp_series = series_generator(10, exp_coefficient);
exp_series(2);
// Other way to call: series_generator(10, n => 1 / factorial(n)
```

Answer (contributed by Xiu Wen - T08D)

Studio Sheet (and Photo Taking)

Additional Material

Church Numerals - Functional Expressionism Quest

- Peano arithmetic - how to describe the natural numbers
- A set of axioms / postulates: (here, $\mathbb{N}$ denotes the set of natural numbers)
    1. 0 is a natural number
    2. $\forall x \in \mathbb{N}, x = x$ (equality is reflexive)
    3. $\forall x, y \in \mathbb{N}, x = y \rightarrow y = x$ (equality is symmetric)
    4. $\forall x, y, z \in \mathbb{N}, \Big(x = y \text{ and } y = z\Big) \Rightarrow x = z$ (equality is transitive)
    5. $\forall a, b; \Big(b \in \mathbb{N} \text{ and } a = b\Big) \Rightarrow a \in \mathbb{N}$ (closure under equality)
    6. $\forall n \in \mathbb{N}, S(n) \in \mathbb{N}$ where S is the successor function (closure under S)
    7. $\forall m, n \in \mathbb{N}, m = n \iff S(m) = S(n)$ (S is an injection)
    8. $\forall n \in \mathbb{N}, S(n) = 0$ is **false**. (No natural number whose successor is 0)
    9. If K is a set such that $0 \in K$ and $\forall n \in \mathbb{N}, n \in K \Rightarrow S(n) \in K$ (induction)

Church Numerals - Functional Expressionism Quest

In lambda calculus, we can represent natural numbers too.
We define 0 as `f => x => x`
1 is defined as `f => x => f(x)`
2 is defined as `f => x => f(f(x))`
... n is defined as `f => x => f(f(f(...f(x)...)))`

The successor function is defined as applying `f` one more time.
In other words, `n => f => z => f(n(f)(z));`

Church Numerals - Functional Expressionism Quest

How do we decrement stuff? [Predecessor function]
Rules: pred(0) $=$ 0 (can't go lower than that), otherwise pred(n) $=$ m iff succ(m) $=$ n.

To implement: [try google, I'm too tired to make these slides]