Functional Programming

(Currying and Uncurrying)

Professor: Suman Saha

Multi-Arguments Functions

```
(define (add m n) (+ m n))
```

add: Int, Int $\rightarrow$ Int

Pass multiple parameters as a list?

```
(add '(1 2))
```

✗

`add` takes 2 parameters

```
(add 1 2)
```

✓

Multi-Arguments Functions

```
(define (add m n) (+ m n))
```

$\text{add: Int, Int} \rightarrow \text{Int}$

- Add 2 to each element in a list?
- map f l: applies function f to each element of list

```
(map (add 2) '(1 2 3))
```

?

Multi-Arguments Functions

```
(define (add m n) (+ m n))
```

add: Int, Int → Int

- Add 2 to each element in a list?
- map f l: applies function f to each element of list

```
(map (add 2) '(1 2 3))
```

`add` takes two parameters

A multi-argument function can only be used when all parameters are ready!

Currying: Every function is treated as taking at most one parameter

```
(define (add m n) (+ m n))
```

add: Int, Int → Int

Curried version

```
(define (addN n) (lambda (m) (+ m n)))
```

addN: Int → (Int → Int)

also written as: Int → Int → Int (right associative)

Uncurried VS Curried

Uncurried version

```
(define (add m n)
        (+ m n))
```

add: $\text{Int}, \text{Int} \rightarrow \text{Int}$

```
(add 2 3)
```

✓

```
(map (add 2)
     '(1 2 3))
```

✗

Curried version

```
(define (addN n)
  (lambda (m) (+ m n)))
```

addN: $\text{Int} \rightarrow (\text{Int} \rightarrow \text{Int})$

```
((addN 2) 3)
```

✓

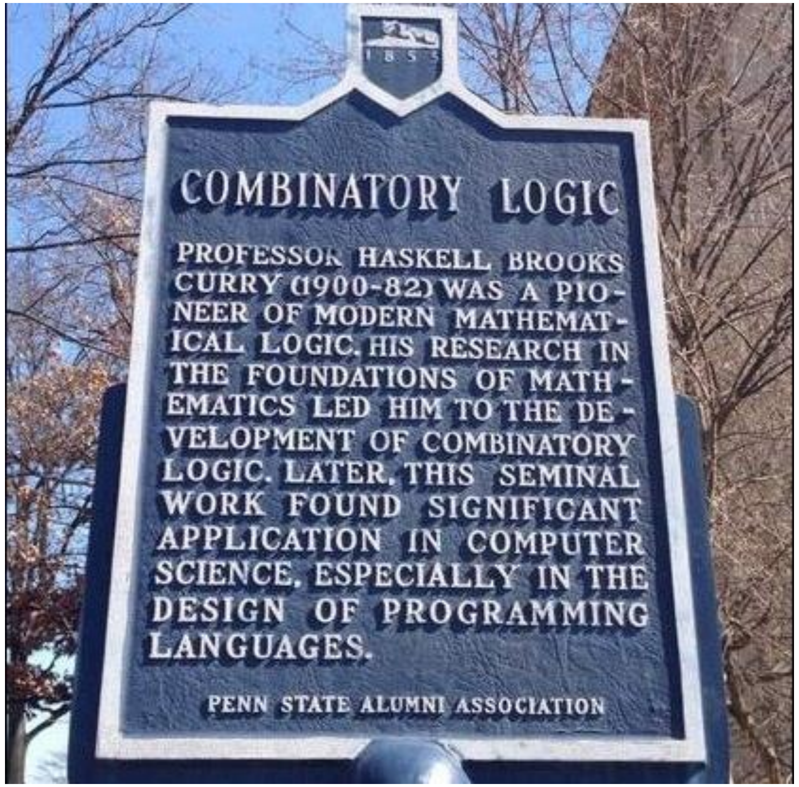
```
(map (addN 2)
     '(1 2 3))
```

✓

Curried form allows partial evaluation

Currying

Haskell B. Curry
Penn State 1929-1966



Outside of McAllister Building

Currying

In terms of lambda calculus, the curried function of

$\lambda x_1 x_2 \ldots x_n . e$ is $\lambda x_1 . (\lambda x_2 . (\ldots(\lambda x_n . e)\ldots))$

```
(define (curry2 f)
  (lambda (x)
    (lambda (y)
      (f x y))))
```

```
(define (curry3 f)
  (lambda (x)
    (lambda (y)
      (lambda (z)
        (f x y z)))))
```

Partial Evaluation

A function is evaluated with one or more of the leftmost actual parameters

$((\text{curry2 add})\ 2)$ is a partial evaluation of add

We can think it as a temporary result, in the form of a function

Partial Evaluation

A function is evaluated with one or more of the leftmost actual parameters

```
(map ((curry2 add) 2) '(1 2 3))
```

✓

add: $\lambda x\ y.\,(+\ x\ y)$

(curry2 add): $\lambda x.\,\lambda y.\,(+\ x\ y)$

$\big($(curry2 add) 2$\big)$: $\lambda y.\,(+2\ y)$

(map ((curry2 add)2)) ’(1 2 3): ’(3 4 5)

Uncurrying

In terms of lambda calculus, the uncurried function of

$\lambda x_1 . (\lambda x_2 .(\ldots(\lambda\ x_n .e)\ldots))$ is $\lambda x_1 x_2 \ldots x_n .e$

```
(define (uncurry2 f)
  (lambda (x y)
      ((f x) y)))
```

```
(define (uncurry3 f)
  (lambda (x y z)
        (((f x) y) z)))
```