アルゴリズムとデータ構造入門
1.手続きによる抽象の構築
1.2 プログラムの構築

奥 乃 博
大学院情報学研究科知能情報学専攻
知能メディア講座 音声メディア分野
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/07/IntroAlgDs/
okuno@i.kyoto-u.ac.jp

TAの居室は10号館4階奥乃1研，2研
TAのページがオープン，質問箱もあります

http://winnie.kuis.kyoto-u.ac.jp/~fukubaya/AlgDsWiki/

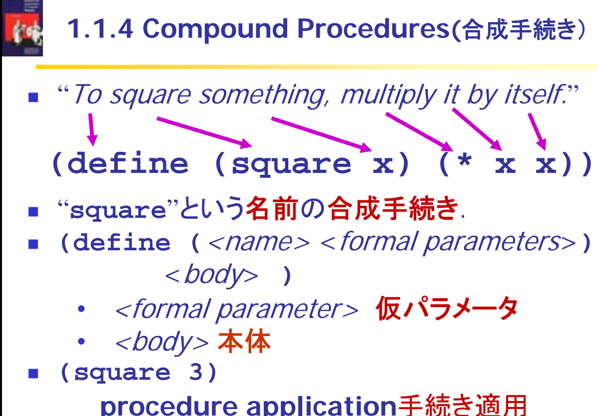
「アルゴリズムとデータ構造入門」中間試験

- 日時：10月30日（火）3限（13:00~14:30）
- 場所：8号館共同2（情報学科1回生）
  共同5（上記以外）
- 出題範囲：
  - 教科書1章（10月23日までの講義）
  - 具体的な範囲は10月23日の講義で指示
- 学生証を忘れないこと
- 遅刻・早退等は定期試験実施要領に準ずる

担当教員： 奥乃 博

10月16日・本日のメニュー

- 1-1-5 The Substitution Model for Procedure Application （復習）
- 1-1-6 Conditional Expressions and Predicates （復習）
- 1-1-7 Example: Square Roots by Newton's Method
- 1-1-8 Procedures as Black-Box Abstractions
- 1.2.1 Linear Recursion and Iteration（復習）
- 1.2.2 Tree Recursion （復習）
- 1.2.3 Orders of Growth



1-1-5 The Substitution Model for Procedure Application (置換モデル)

- **Vocabulary (語彙)** ⇒Primitives
- **Syntax (構文)** ⇒means of abstractions
- **Semantics (意味)** ⇒ Viewing the rules of evaluation from a computational perspective (計算という観点からの評価法)

- 手続き適用の評価法として「置換モデル」

5

置換モデルの例による説明

```
(define (square x) (* x x))
(define (sum-of-squares x y)
   (+ (square x) (square y)) )
(define (f a) (sum-of-squares (+ a 1) (* a 2)))
(f 5)
     (sum-of-squares (+ a 1) (* a 2)) に a = 5 を適用
(sum-of-squares (+ 5 1) (* 5 2))  で a を 5 で置換(代入)
     (+ (square x) (square y))  に x = 6, y = 10 を適用
(+ (square 6) (square 10)) で x を 6 で, y を 10 で置換
     (* x x) に x = 6, (* x x) に x = 10 を適用
(+ (* 6 6) (* 10 10)) で x を 10 で置換
(+ 36 100)
136
```

6

Applicative order vs. normal order
(適用順序と正規順序)

- 今見てみた置換モデルの評価順序：
  「適用順序(作用順序, Applicative order)」
- 別の順序:「正規順序(normal-order)」：
  仮パラメータをすべて置換してから，簡約する．

```
1.  (f 5)
2.       ((sum-of-squares (+ a 1) (* a 2)) 5)
3.  (sum-of-squares (+ 5 1) (* 5 2))
4.       ((+ (square x) (square y)) (+ 5 1) (* 5 2))
5.  (+ (square (+ 5 1)) (square (* 5 2))
6.       (+ ((* x x) (+ 5 1)) ((* x x) (* 5 2))
7.  (+ (* (+ 5 1) (+ 5 1)) (* (* 5 2) (* 5 2)))
8.  (+ (* 6 6) (* 10 10))
9.  (+ 36 100)
10. 136
```

2回同じものを計算

7

1.1.6 Condtional Expressions and Predicates (条件式と述語)

- 絶対値をcase analysis (場合分け)で定義

$$|x| = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -x & \text{if } x < 0 \end{cases}$$

cond, if の実行は、置換モデルで、特別扱い
Special form(特殊形式)

```
1.(define (abs x)
    (cond ((> x 0) x)
          ((= x 0) 0)
          ((< x 0) (- x)) ))
```

if : Syntax sugar

糖衣

```
2.(define (abs x)
    (cond ((< x 0) (- x))
          ((>= x 0) x) ))
```

```
(or (> x 0) (= x 0))
```

```
3.(define (abs x)
    (cond ((< x 0) (- x))
          (else x) ))
```

```
4.(define (abs x)
    (if (< x 0)
        (- x)
        x ))
```

8

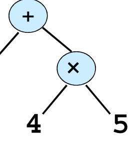
1.1.6 Condtional Expressions and Predicates (条件式と述語)

- 条件式の一般形; cond は特殊形式 (special form)
- (cond ($<p_1>$ $<e_{11}>$ …$<e_{1m}>$)
  ($<P_2>$ $<e_{21}>$…$<e_{2k}>$)
  …
  ($<p_n>$ $<e_{n1}>$…$<e_{np}>$))
- 式の対 ($<p>$ $<e>$ … $<e>$) : 節(clause)
- $<p>$: 述語(predicate)
- 述語の値: true (#t) か false (#f).
- $<e>$: 帰結式(consequent expression)
- 特別の$<p>$: else (#t を返す)
- 節の評価は, $<p>$が#tなら$<e>$が順に評価される.
- 一旦述語が#tを返すと, それ以降の節は評価されない.

9

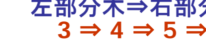
Ex.1.2 前置記法(prefix notation)

- 式 （演算子　　被演算子　…）
  operator　operands
- 式の記法
  - 前置記法 （prefix notation， Pollish notation, ポーランド記法)
    ＋ 3 * 4 5
  - 中置記法 （infix notation)
    3 ＋ (4 * 5)
  - 後置記法 （postfix notation, reverse Pollish notation、逆ポーランド記法)
    3 4 5 * ＋
- 木表現はどれも同じ



木の辿り方から3つの記法への変換

- 木の辿り方
  - 前順走査 （pre-order traversal)
    ノード⇒左部分木⇒右部分木
    ＋ ⇒ 3 ⇒ * ⇒ 4 ⇒ 5
  - 間順走査 （in-order tr.)
    左部分木⇒ノード⇒右部分木
    3 ⇒ ＋ ⇒ 4 ⇒ * ⇒ 5
  - 後順走査 （post-order tr.)
    左部分木⇒右部分木⇒ノード
    3 ⇒ 4 ⇒ 5 ⇒ * ⇒ ＋



+
×
3
4
5

Javaプログラムのデモ

http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/07/IntroAlgDs/

Ex.1.3 Define a procedure that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.

```
(define (sum-of-two-sq x y z)
   (cond ((> x y)
          (cond ((> y z)
                 (sum-of-squares x y) )
                (else (sum-of-squares x z)) ))
         ((> x z) (sum-of-squares y x))
         (else (sum-of-squares y z)) ))
```

```
(define (sum-of-two-sq x y z)
   (if (> x y)
       (if (> y z)
           (sum-of-squares x y)
           (sum-of-squares x z) )
       (if (> x z)
           (sum-of-squares y x)
           (sum-of-squares y z) )))
```

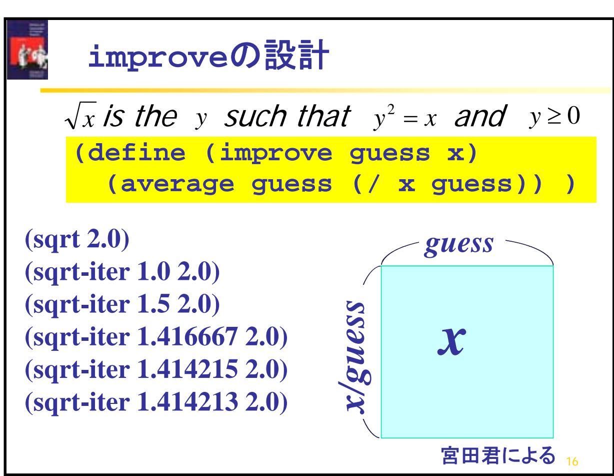
1.1.7 Square Root by Newton's Method

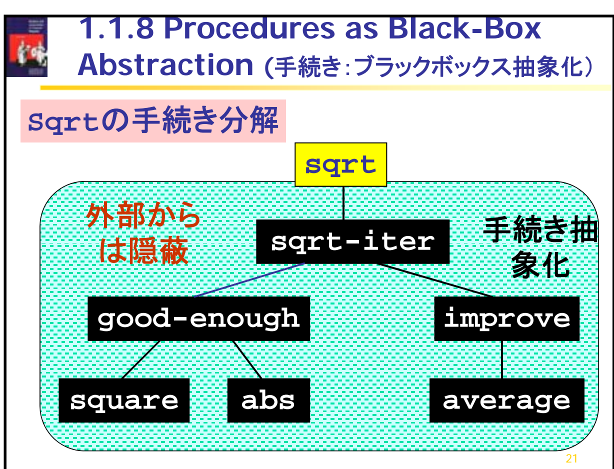
$\sqrt{x}$ *is the* y *such that* $y^2 = x$ *and* $y \geq 0$

Recursive (再帰的)

```
(define (sqrt-iter guess x)
  (if (good-enough? guess x)
      guess
      (sqrt-iter (improve guess x) x) ))

(define (improve guess x)
  (average guess (/ x guess)) )

(define (average x y)
  (/ (+ x y) 2) )

(define (good-enough? guess x)
   (< (abs (- (square guess) x)) 0.001)

(define (sqrt x) (sqrt-iter 1.0 x))
```

15





手続き抽象化の効用 Square の定義

1.内部実装(implementation)の隠蔽

- `(define (square x) (* x x))` x^2
- `(define (square x)`
  `(exp (double (log x))) )` $e^{2\ln x}$
  `(define (double x) (+ x x))`

2.局所名(local names)の隠蔽

- `(define (square x) (* x x))`
- `(define (square y) (* y y))`

22

束縛変数 (bound variables)と自由変数(free variables)

bound (束縛)

```
(define (sqrt-iter guess x)
  (if (good-enough? guess x)
      guess
      (sqrt-iter (improve guess x) x) ))

(define (good-enough? guess x)
   (< (abs (- (square guess) x)) 0.001)

(define (good-enough? v target)
   (< (abs (- (square v) target)) 0.001)
```

- 束縛変数:仮パラメータは手続きで束縛
- 自由変数:束縛・captureされていない
- 有効範囲(scope)変数の束縛されている式の範囲

23

Block Structure(ブロック構造):x の scope(有効範囲)は

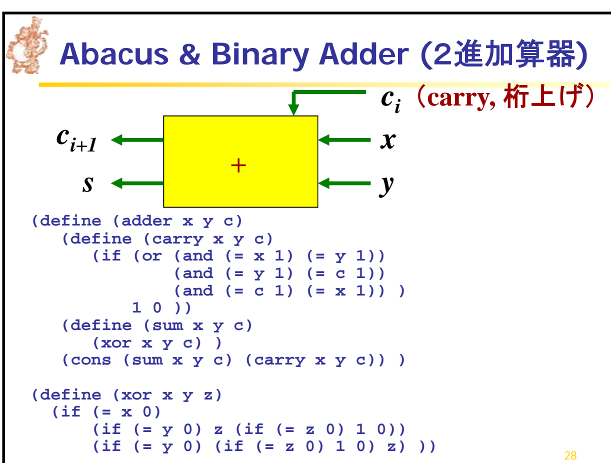
```
(define (sqrt x)
  (define (improve guess x)
    (average guess (/ x guess)) )
  (define (good-enough? guess x)
    (< (abs (- (square guess) x)) 0.001) )
  (define (sqrt-iter guess x)
    (if (good-enough? guess x)
        guess
        (sqrt-iter (improve guess x) x) ))
  (sqrt-iter 1.0 x) )
```

静的有効範囲(lexical scoping)

10月16日・本日のメニュー

- 1-1-5 The Substitution Model for Procedure Application （復習）
- 1-1-6 Conditional Expressions and Predicates （復習）
- 1-1-7 Example: Square Roots by Newton's Method
- 1-1-8 Procedures as Black-Box Abstractions
- **Intermission**
- 1.2.1 Linear Recursion and Iteration（復習）
- 1.2.2 Tree Recursion （復習）
- 1.2.3 Orders of Growth

26



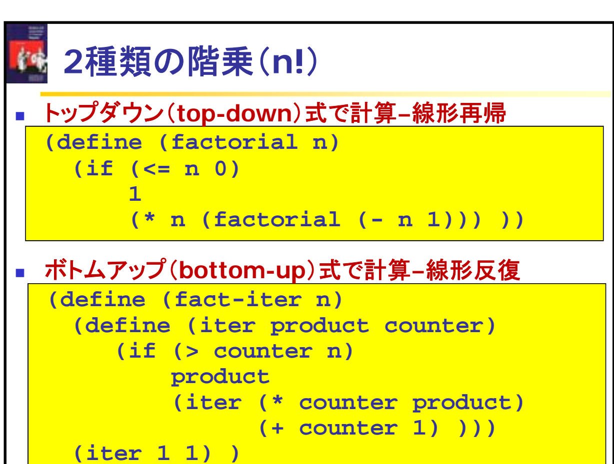
宿題1：アッカーマン関数の値

```
(define (ackermann m n)
      (if (= m 0)
          (+ n 1)
          (if (= n 0)
              (ackermann (- m 1) 1)
              (ackermann (- m 1)
                    (ackermann m (- n 1))))
```

1. (ackermann 0 n) ≡
2. (ackermann 1 n) ≡
3. (ackermann 2 n) ≡
4. (ackermann 3 n) ≡
5. (ackermann 4 n) ≡

10月16日・本日のメニュー

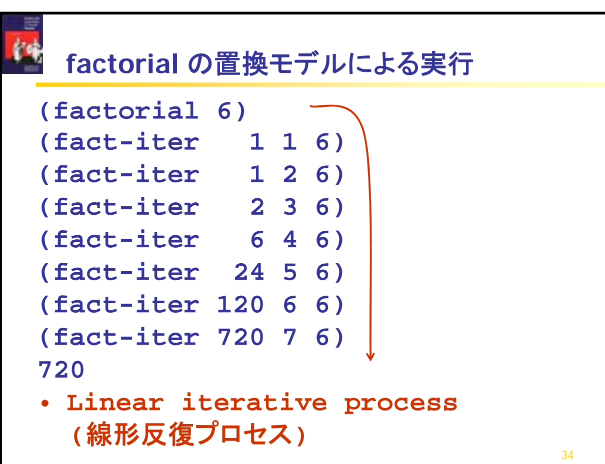
- 1-1-5 The Substitution Model for Procedure Application （復習）
- 1-1-6 Conditional Expressions and Predicates （復習）
- 1-1-7 Example: Square Roots by Newton's Method
- 1-1-8 Procedures as Black-Box Abstractions
- Intermission
- **1.2.1 Linear Recursion and Iteration（復習）**
- 1.2.2 Tree Recursion （復習）
- 1.2.3 Orders of Growth



factorial の置換モデルによる実行

```
(factorial 6)
(* 6 (factorial 5))
(* 6 (* 5 (factorial 4)))
(* 6 (* 5 (* 4 (factorial 3))))
(* 6 (* 5 (* 4 (* 3 (factorial 2)))))
(* 6 (* 5 (* 4 (* 3 (* 2 (factorial 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 (* 1 (factorial 0)))))))
(* 6 (* 5 (* 4 (* 3 (* 2 (* 1 1))))))
(* 6 (* 5 (* 4 (* 3 (* 2 1)))))
(* 6 (* 5 (* 4 (* 3 2))))
(* 6 (* 5 (* 4 6)))
(* 6 (* 5 24))
(* 6 120)
720
```

33



Tail recursion の補足説明

```
(define (f n)
  (if (<= n 0)
      1
      (* (f (- n 1)) n) ))
```

- このプログラムは次の翻訳
  n! = (n-1)! * n
- 下記のfactorialとの違いは

```
(define (factorial n)
    (if (<= n 0)
        1
        (* n (factorial (- n 1))) ))
```

39

factの置換モデルによる実行

```
(f 6)
(* (f 5) 6)
(* (* (f 4) 5) 6)
(* (* (* (f 3) 4) 5) 6)
(* (* (* (* (f 2) 3) 4) 5) 6)
(* (* (* (* (* (f 1) 2) 3) 4) 5) 6)
(* (* (* (* (* (* (f 0) 1) 2) 3) 4) 5) 6)
(* (* (* (* (* (* 1 1) 2) 3) 4) 5) 6)
(* (* (* (* (* 1 2) 3) 4) 5) 6)
(* (* (* (* 2 3) 4) 5) 6)
(* (* (* 6 4) 5) 6)
(* (* 24 5))
(* 120)
720
```

40

Procedure（手続き）vs. Process（プロセス）

- 手続きが再帰的とは、構文上から定義。
  自分の中で自分を直接・間接に呼び出す。
- 再帰的手続きの実行
  - 再帰プロセスで実行
  - 反復プロセスで実行
- 線形再帰プロセスは線形反復プロセスに変換可能
  「tail recursion（末尾再帰的）」
- 再帰プロセスでは、deferred operation用にプロセスを保持しておく必要がある
  ⇒ スペース量が余分にいる。
- Scheme のループ構造はsyntactic sugar
  - do, repeat, until, for, while

42

Ex.1.10 Ackermann Function

```
(define (A x y)
  (cond ((= y 0) 0)
        ((= x 0) (* 2 y))
        ((= y 1) 2)
        (else (A (- x 1)
                 (A x (- y 1)) ))))
```

- Ackermann関数は線形再帰ではない！

```
(define (Ack m n)
  (cond ((= m 0) (+ n 1))
        ((= n 0) (Ack (- m 1) 1))
        (else (Ack (- m 1)
                   (Ack m (- n 1)) ))))
```

43

10月16日・本日のメニュー

- 1-1-5 The Substitution Model for Procedure Application （復習）
- 1-1-6 Conditional Expressions and Predicates （復習）
- 1-1-7 Example: Square Roots by Newton's Method
- 1-1-8 Procedures as Black-Box Abstractions
- **Intermission**
- **1.2.1 Linear Recursion and Iteration（復習）**
- **1.2.2 Tree Recursion （復習）**
- 1.2.3 Orders of Growth

44

2種類のFibonacci 関数

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1))
                 (fib (- n 2)) ))))
```

- トップダウン(top-down)式に計算 – 木構造再帰

```
(define (fib n)
   (define (iter a b count)
       (if (= count 0)
           b
           (iter (+ a b) a (- count 1)) ))
   (iter 1 0 n) )
```

- ボトムアップ(bottom-up)式に計算 – 線形再帰だが、線形反復プロセス

46

1.2.2 Fibonacci – Tree Recursion

fib 5
fib 4
fib 3
fib 3
fib 2
fib 2
fib 1
fib 2
fib 1
fib 1
fib 0
fib 1
fib 0
fib 1
fib 0
1
1
0
1
0
1
0
1

木構造再帰

48

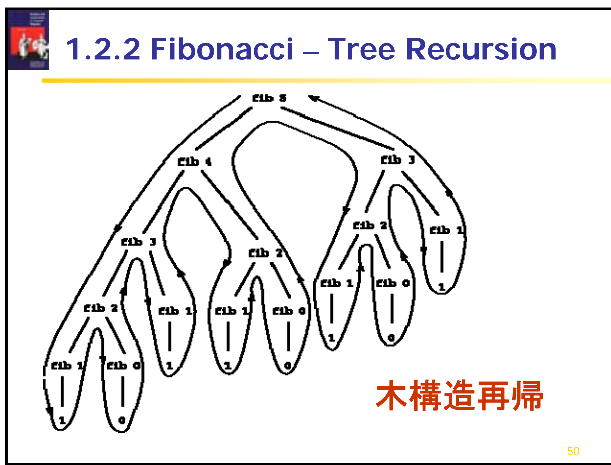
fib-iter の置換モデルによる実行

```
(fib-iter 6)
  (iter  1 0 6)
  (iter  1 1 5)
  (iter  2 1 4)
  (iter  3 2 3)
  (iter  5 3 2)
  (iter  8 5 1)
  (iter 13 8 0)
8
```

- Linear iterative process
  （線形反復プロセス）

49

1.2.2 Fibonacci – Tree Recursion



木構造再帰

50

Fibの呼ばれる回数　C(n)

- C(n):n　に対して　`fib`　の呼ばれる回数
- C(0)=C(1)=1
- n≧2 に対して　C(n)=C(n-1)+C(n-2)+1
- C(2)=3, C(3)=5, C(4)=9, C(5)=15, …
- 一般に、$C(k) > 2^{k/2}$

C(n)　を求める

1. F(n)=C(n)+1 とおくと
2. F(n)=F(n-1)+F(n-2)　for n≧2
3. F(0)=2, F(1)=2

51

Fibの呼ばれる回数 C(n) =F(n)-1

- n≧2 に対して　$C(n)=C(n-1)+C(n-2)+1$
- $F(n)=C(n)+1$ とおくと
- $F(n)=F(n-1)+F(n-2)$　for n≧2
- $F(0)=2, F(1)=2$

さあ、解いてみてください。

Ex. Counting Change

1. 1ドルの両替の方法は何通り？
2. 50セント (half dollar), 25セント (quarter), 10セント (dime), 5セント(nickel), 1セント(penny)

3. 一般化：　分割数を求める

57

n種類の硬貨，金額aの両替の場合の数は

- 使える硬貨をある順番で並べておくと
- n種類の硬貨で金額aの両替の場合の数は
  1. 先頭の種類を除いたすべての硬貨を使って金額aを両替する場合の数
     ＋
  2. 先頭の種類の硬貨(額面d)とすると，a−dの額を全n種の硬貨を使って両替する場合の数
  3. 初期値：a=0の時1，a<0の時かn=0の時0

分割統治法(divide-and-conquor)

58

Ex. Counting Change

```
(define (count-change amount)
  (cc amount 5) )

(define (cc amount kinds-of-coins)
  (cond ((= amount 0) 1)
        ((or (< amount 0) (= kinds-of-coins 0)) 0)
        (else (+ (cc amount (- kinds-of-coins 1))
                 (cc (- amount (first-denomination
                                   kinds-of-coins))
                     kinds-of-coins )))))

(define (first-denomination kinds-of-coins)
  (cond ((= kinds-of-coins 1) 1)
        ((= kinds-of-coins 2) 5)
        ((= kinds-of-coins 3) 10)
        ((= kinds-of-coins 4) 25)
        ((= kinds-of-coins 5) 50) ))
```

60

10月16日・本日のメニュー

- 1-1-5 The Substitution Model for Procedure Application（復習）
- 1-1-6 Conditional Expressions and Predicates（復習）
- 1-1-7 Example: Square Roots by Newton's Method
- 1-1-8 Procedures as Black-Box Abstractions
- Intermission
- 1.2.1 Linear Recursion and Iteration（復習）
- 1.2.2 Tree Recursion （復習）
- **1.2.3 Orders of Growth**

62

1.2.3 Order of Growth

R(n) は、ステップ数あるいはスペース量

- *R(n)* が $\Theta(\mathbf{f(n)})$ $k_1 f(n) \geq R(n) \geq k_2\ f(n)$
- *R(n)* が $\mathrm{O}(\mathbf{f(n)})$ $R(n) \leq k\ f(n)$
  上限
- *R(n)* が $\Omega(\mathbf{f(n)})$ $R(n) \geq k\ f(n)$
  下限

For all $n > n_0$

63

Order of Growth R(n)の例

- $\Theta(1)$: constant growth
- $\Theta(n)$: linear growth
- $\Theta(b^n)$: exponential growth
- $\Theta(\log n)$: logarithmic growth
- $\Theta(n^m)$: power law growth

64

ギリシャ文字

Α	α	alpha	Ν	ν	nu
Β	β	beta	Ξ	ξ	xi
Γ	γ	gamma	Ο	ο	omicron
Δ	δ	delta	Π	π	pi
Ε	ε	epsilon	Ρ	ρ	rho
Ζ	ζ	zeta	Σ	σ	sigma
Ψ	η	eta	Τ	τ	tau
Θ	θ	theta	Υ	υ	upsilon
Ι	ι	iota	Φ	φ	phi
Κ	κ	kappa	Χ	χ	chi
Λ	λ	lambda	Ψ	ψ	psi
Μ	μ	mu	Ω	ω	omega

80