# ソフトウェア基礎論

## アルゴリズムの正当性

情報工学科
田中哲雄

神奈川工科大学
KANAGAWA INSTITUTE OF TECHNOLOGY

# 今日の講義の目標と内容

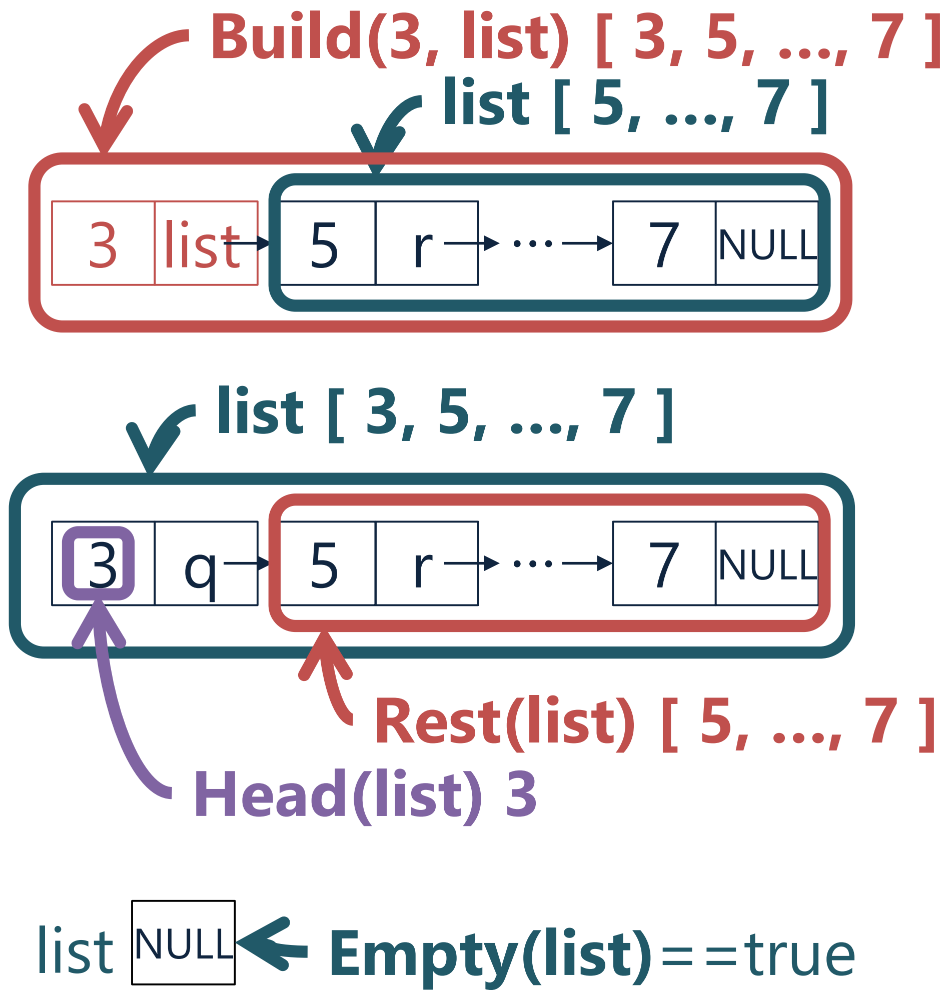
- 目標
  - アルゴリズムの正当性について理解する

- 内容
  1. リストの復習
  2. 数学的帰納法の復習
  3. アルゴリズムの正当性
  4. 構造的帰納法
  5. 再帰的データ構造を扱うアルゴリズムの正当性

# 1. リストの復習 (1) 基本演算

## ■ 基本演算

- **Build** ：リストを作成する
- **Head** ：リストの先頭の要素を得る
- **Rest** ：残りのリストを得る
- **Empty** ：リストが空か




## ■ 基本演算の性質

- Head(Build(x,q)) = x
- Rest(Build(x,q)) = q

# 1. リストの復習 (1) 基本演算

演習

- 要素が順に1, 2, 3, 4のリストを[1,2,3,4]と書くことにする. 次の式が表すリストを記せ.
  1. Build(5, NULL)
     [5]
  2. Build(3, Build(4, Build(5, NULL)))
     [3,4,5]
  3. Rest(Build(3, Build(4, Build(5, NULL))))
     [4,5]
- 次の式の値を求めよ. 真理値はtrueとfalseで記せ
  1. Head(Build(3, Build(4, Build(5, NULL)))) 3
  2. Empty(Build(3, Build(4, Build(5, NULL)))) false
  3. Empty(NULL) true

# 1. リストの復習 (1) 基本演算

演習

- 次のリストを表す式をBuildを用いて記せ
  1. [ 10 ]

     Build(10, NULL)
  2. [ 27, 10 ]

     Build(27, Build(10, NULL))
  3. [ 53, 27, 10 ]

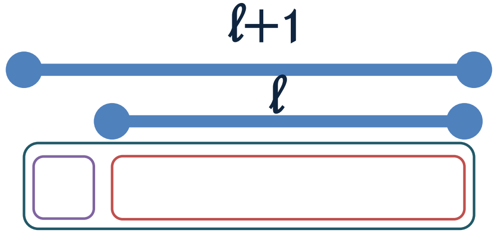
     Build(53, Build(27, Build(10, NULL)))
- 上記3のリストをL3とする. このときL3から2番目の値(27)を取り出す式をL3, Rest, Headを用いて記せ　Head(Rest(L3))

# 1. リストの復習 (2)リストの長さ

- リストの長さを求めるアルゴリズム
  - リストが空なら 長さは0
  - 空でなければ 残りのリストの長さ ＋ 1

```
int Length (IntList ls) {
    if (Empty(ls)) return   0    ;
    return Length(Rest(ls)) +1  ;
}
```

- 長さに関する性質
  - Build(x,p)の長さは, pの長さ+1
    - ◆ Length(Build(x,p)) = Length(p) + 1
  - pが空でなければ, pの長さは残りのリストの長さ+1
    - ◆ Length(p) = Length(Rest(p))+1



# 2. 数学的帰納法の復習 (1)定義

- **数学的帰納法** (mathematical induction)
  - Pを正の整数(あるいは非負整数)nに関する**命題**とするとき，任意のnについてPが成り立つことを証明するためには，次の(1)(2)を示せばよい

> (1) n=1のとき(あるいは0のとき)，Pが成り立つ
>
> (2) n=kのときPが成り立つと**仮定すれば**，
> n=k+1のときもPが成り立つ

  - このような証明法を数学的帰納法という
  - (1)を **初期ステップ** ，(2)を **帰納ステップ** と呼ぶ
- **命題**：真偽を判断できる文や式
  - $1 + 2 + \cdots + n = n(n+1)/2$
  - $1 + 3 + \cdots + (2n-1) = n^2$

# 2. 数学的帰納法の復習 (2) 妥当性

- 数学的帰納法の妥当性
  - 初期ステップより
    n = 1 のとき成り立つ
  - 帰納ステップより
    n = 1 のとき成り立つので，n = 2 のとき成り立つ
    n = 2 のとき成り立つので，n = 3 のとき成り立つ
    n = 3 のとき成り立つので，n = 4 のとき成り立つ
    n = 4 のとき成り立つので，n = 5 のとき成り立つ
    n = 5 のとき成り立つので，n = 6 のとき成り立つ
    n = 6 のとき成り立つので，n = 7 のとき成り立つ
    ...

# 2. 数学的帰納法の復習 (3) 例

- $1 + 2 + \cdots + n = \frac{n*(n+1)}{2}$ … 式1 の証明

(1) $n = 1$ のとき,

左辺 $= 1 = \frac{1*(1+1)}{2} =$ 右辺 となり成り立つ(初期ステップ)

(2) $n = k$ のとき成り立つと仮定すると

$1 + 2 + \cdots + k = k*(k+1)/2$ である

$n = k + 1$のとき

左辺 $= 1 + 2 + \cdots + k + (k + 1) = \frac{k(k+1)}{2} + (k + 1)$

$= \frac{k(k+1)+2(k+1)}{2} = \frac{(k+1)(k+2)}{2} =$ 右辺

となり成り立つ(帰納ステップ)

(1)(2)より, 式1は全てのnについて成り立つ

# 2. 数学的帰納法の復習 (3) 例

演習

■ $1 + 3 + \cdots + (2n - 1) = n^2$ ･･･ 式2 の証明

(1) $n = 1$ のとき，　（n2をn**2と書くことにする）

左辺 = 1 = 1**2 = 右辺 となり成り立つ(初期ステップ)

(2) $n = k$ のとき成り立つと仮定すると

1 + 3 + ･･･ + ( 2k - 1 ) = k**2 である

$n = k + 1$のとき

左辺 = 1 + 3 + ･･･ + ( 2k - 1 ) + ( 2(k+1) -1 )

= k**2 + 2k + 1

= (k+1)**2 = 右辺

となり成り立つ(帰納ステップ)

(1)(2)より，式2は全てのnについて成り立つ

# 3. アルゴリズムの正当性

## ■ アルゴリズムの正当性

- アルゴリズムが停止するときには，必ず正しい結果を出力するという性質を **部分正当性** という
- この性質を持つアルゴリズムは **部分的に正しい** という
- アルゴリズムが（想定された）任意の入力に対して必ず停止するという性質を **停止性** という
- アルゴリズムが**部分的に正しく**，かつ，**必ず停止する**という性質を **正当性** という
- このような性質をもつアルゴリズムは **全域的に正しい** という

**正当性 ＝ 部分正当性 ∧ 停止性**

# 3. アルゴリズムの正当性 (1)部分正当性

- 正整数nの階乗を求めるアルゴリズムfactorialの部分正当性

```
int factorial(int n) {
    if ( n == 1 ) return 1;              ①
    else return factorial(n-1) * n;     ②
}
```

- 数学的帰納法により factorial(n) = n! を示す

(1) n=1のとき（factorial(1)=1! を示す）

左辺 $\underbrace{=}_{①}$ 1 $\underbrace{=}_{階乗の定義}$ 1! = 右辺

(2) n=kのとき成り立つと仮定すると factorial(k)=k!である

n=k+1のとき（factorial(k+1)=(k+1)! を示す）

左辺 $\underbrace{=}_{②}$ factorial(k)*(k+1) $\underbrace{=}_{帰納法の仮定}$ k!*(k+1) $\underbrace{=}_{階乗の定義}$ (k+1)! = 右辺

演習

# 3. アルゴリズムの正当性 (1)部分正当性

- 累乗(mのn乗)を求めるアルゴリズムpowerの部分正当性

```
int power(int m, n) {
    if ( n == 0 ) return 1;            ①
    return power(m, n-1) * m;          ②
}
```

- 数学的帰納法により power(m, n) = $m^n$ を示す

(1) n=0のとき （power(m,0) = $m^0$ を示す）

左辺 $\underbrace{=}_{①}$ 1 $\underbrace{=}_{\text{累乗の定義}}$ $m^0$ = 右辺

(2) n=kのとき成り立つと仮定するとpower(m, k)=$m^k$である

n=k+1のとき（power(m, k+1)=$m^{k+1}$ を示す）

左辺 $\underbrace{=}_{②}$ power(m, k) ∗ m $\underbrace{=}_{\text{帰納法の仮定}}$ $m^k$ ∗ m $\underbrace{=}_{\text{累乗の定義}}$ $m^{k+1}$ = 右辺

# 3. アルゴリズムの正当性 (2)停止性

- 停止性を証明するには,
  - ある **下限値** で押さえられていて, かつ,
  - 再帰呼び出しする度に値が **減少** するような式を見つける
    **上限値** で押さえられていて **増加** するような式でもよい
  - このような式があれば, いつかは下限値に到達し, アルゴリズムは必ず停止する
- 階乗を求めるアルゴリズム
  - n　は下限値　1　で押さえられていて
  - 再帰呼び出しする度に必ず 1　ずつ減少するので,
    いつかは 1　に到達し, 停止する

# 3. アルゴリズムの正当性 (2)停止性

演習

- xとy(ただし x>=y)の最大公約数をもとめるアルゴリズム GCD(x, y)を完成させよ

ヒント: ユークリッドの互除法

x >= y である非負整数 x,y の最大公約数は

- ◆ y==0 ならば x である
- ◆ y>0 ならば y と x%y の最大公約数と等しい

```
int GCD(int x, int y) { // 最大公約数を求める
  if ( y == 0 ) return    x
  return GCD( y, x%y ) ;
}
```

# 3. アルゴリズムの正当性 (2)停止性

演習

- 前スライドのアルゴリズムGCD(a,b)の停止性の証明を完成させよ
  - y は 下限値 0 でおさえられていて,
  - x%y　はyより小さいので, y は再帰呼び出しを1回実行する度に 少なくとも1以上減少し , いつかは 0 に到達する
  - 従って, アルゴリズムは停止する

# 4. 構造的帰納法 (1)数学的帰納法 再考

## ■ 自然数(非負整数)の 構成子

```
unsigned int zero() { return 0; } //ゼロ
unsigned int suc(unsigned int n) { return n+1; } //次の自然数
```

- zeroとsucを組み合せることで，任意の自然数を表現する(構成する)ことができる．3は suc(suc(suc(zero())))
- このような基本演算を構成子(コンストラクタ)という

## ■ zeroとsucを使った数学的帰納法

- 任意の自然数(非負整数)nについて命題Pが成り立つことを証明するには，次の(1),(2)を示せばよい

(1) n = zero() のとき，Pが成り立つ

(2) n = k のときPが成り立つと仮定すれば，
n = suc(k) のときもPが成り立つ

# 4. 構造的帰納法 (2)

- **数学的** 帰納法は **自然数** に関する命題の証明
- **構造的** 帰納法は **再帰的データ構造** に関する命題の証明
  - 自然数の構成子(zero, suc)の代わりに, リストや二分木の構成子を考える
- リストの **構成子** ： NULL と Build
  - 全てのリストは Build(x1, Build(x2, ･･･ ,Buid (xn, NULL)･･･)) の形をしている

slide5

# 4. 構造的帰納法 (3)

- リストに関する構造的帰納法
  - 任意のリストpについて命題Pが成り立つことを証明するには, 次の(1),(2)を示せばよい

(1) p = NULL のとき, Pが成り立つ

(2) p = q のときPが成り立つと仮定すると,

p = Build(x, q) のときもPが成り立つ

# 2. 再帰的データ構造を扱うアルゴリズムの正当性 (1) リストの和

演習

- リストの和を求めるアルゴリズム
  - リストの和を求めるアルゴリズムを完成させよ
  - リストの和 ＝
    - リストが空なら0
    - リストが空でなければ 先頭要素 ＋ 残りのリストの和

```
int Sum(IntList ls) {
    if ( Empty(ls) ) return  0 ;                 ①
    else return Head(ls) + Sum(Rest(ls)) ;  ②
}
```

# 2. 再帰的データ構造を扱うアルゴリズムの正当性 (2) リストの和 ー部分正当性ー

■ リストの和を求めるアルゴリズムの部分正当性

- Sum(ls) = lsの和 を構造的帰納法で証明する

(1) 初期ステップ

①　和の定義

左辺 = Sum(NULL) = 0 = NULLの和 = 右辺

演習

(2) 帰納ステップ

p=qのとき成り立つと仮定すると Sum(q) = qの和

Sum(Build(x,q)) = Build(x,q)の和 を証明する

左辺 = Head(Build(x,q))+Sum(Rest(Build(x,q))　②

= x + Sum(q)　基本演算に関する性質

= x + qの和　帰納法の仮定

= Build(x,q)の和　和の定義

= 右辺

slide3

# 2. 再帰的データ構造を扱うアルゴリズムの正当性 (3) リストの和 －停止性－

演習

- リストでは, リストの長さに着目する　slide6
- リストの和を求めるアルゴリズムでは
  - リストの長さ は 下限値 0 でおさえられていて,
  - Rest(ls) の長さは, ls の長さよりも 1 小さいので, リストの長さは, 再帰呼出しを 1回実行する度に, 1 ずつ減少し, いつかは 0 に到達する
  - 従って, アルゴリズムは停止する