# ソフトウェア概論 A/B

数学科 吉開範章, 渡辺俊一 (栗野 俊一)

2010/10/15 ソフトウェア概論

# おしらせ

□ ソフトウェア概論 B は原則、栗野が担当します。

□ 資料

http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2010/soft/soft.html

# 本日の概要:色々なプログラムを作ってみよう

- □ 講義内容
  - ○ 色々なプログラムを作ってみよう ( 8 章 : p.184　p.205 )
    - ▹ 再帰 ( 8-3 節 : p.194　p.197 )
- □ 課題
  - ○ 演習 8-6 (Text p.197)

# 前回までの復習

- □マクロとは
  - ○C のプログラム内の「単語」を他の単語列に置き換える機能
    - ▷コンパイラが C のプログラムを読む前に置き換え(マクロ展開)が行われる
    - ▷cf. cpp : C PreProcesser
  - ○C 言語の機能不足部分を補うために追加された
    - ▷本来 C 言語と独立した機能
  - ○強力だが、強力過ぎて危険な所もある
    - ▷使い方を限定して利用した方がよい
- □マクロ機能の利用例
  - ○定数の定義
    - ▷「マジックナンバー」の追放
  - ○関数形式マクロ
    - ▷テキスト p.184-189
    - ▷引数付きマクロ
  - ○ファイルの取り込み
    - ▷cf. #include<stdio.h>
- □マクロで失敗しないためには
  - ○マクロ展開によってどの様に置き換えがおきるかを考える

# 今週の内容

- 再帰的プログラム
  - 関数の再帰的定義
  - 再帰的関数の振舞い
  - 再帰的関数と数学的帰納法
- 分割統治
  - 「棚上げ」の構造

# 再帰的プログラム

- □ 関数の再帰的定義とは？
  - ○ 関数 f の本体に f が現れる
    - ▷ cf. factrial (sample-001.c)
    - ▷ 言ってみれば「それだけ」なのだが..
    - ▷ 「難しい」と評判(?)
  - ○ 数学の世界では、「(数学的)帰納法」/「帰納的関数」と呼ばれる
    - ▷ 数学では「普通」の考え方
- □ 再帰的でない関数定義
  - ○ 新しい関数を定義する場合は、既定義の関数を利用する
    - ▷ 他の関数を呼ばない関数(スタブ関数)がある (sample-002.c)
    - ▷ 関数の呼び出し回数が有限である事が解る
  - ○ 再帰関数の場合は呼び出し回数を事前に知る事ができない
    - ▷ 繰り返しと同じ構造を持つ

# 再帰的プログラムは難しいか？

- 「再帰」が「難しい」理由？
  - そのそも「再帰の考え方」が解らない
    - 数学科の学生は大丈夫だよね..
    - How to より What is 的な表現
  - 「ブロックの積み上げ」でなくなっている
    - これまでは、既定義の関数を利用して新しい関数を定義する
    - 再帰的定義では、これから定義する関数を利用している
  - 動作に関する不理解
    - 再帰的定義からすぐに、実行順序が予測できない
  - 表現上は「繰り返し」が含まれていないが「繰り返し」の機能
    - まちがえると、無限ループになる
    - 「終了条件抜け」問題

# 数学的帰納法

- □ 数学的帰納法とは
  - ○ ある性質 P が全ての自然数 n で成立する事を示す証明形式
- □ 数学的帰納法による証明
  - ○ n=0 の時に P(n) が成立する事を示す
    - ▹ この部分はプログラミング上では重要(終了条件問題)
  - ○ n=k の時に P(n) が成立する事を利用し、n=k+1 の場合も成立する事を示す
    - ▹ P を証明するのに P を使っている所がミソ(自然数の構造)
    - ▹ 関数定義の場合は、この部分が「再帰的定義」の部分
  - ○ 無限の数の命題をたった二つの記述で示せる強力な手段
    - ▹ P(n) の形の命題は n が自然数なので無限個存在する
    - ▹ 「無限を有限にする」(「繰り返し」を繰り返しを使わず表現する)仕組
- □ 帰納法の例(全ての自然数は偶数か奇数である)
  - ○ 帰納法による証明
    - ▹ n=0) 0 は偶数なので、「偶数か奇数」は正しい
    - ▹ n=k+1) 帰納法の仮定より k は偶数か奇数である
    - ▹ n=k が偶数の時は、n=k+1 は奇数になる/ n=k が奇数の時は、n=k+1 は偶数になる
    - ▹ k が偶数でも奇数でも k+1 は奇数か偶数になる。すなわち、n=k+1 の時も n は奇数か偶数である

# 再帰と帰納法

- 帰納法の考え方
  - 最初の一歩と次の一歩を分けて考える
  - 言わば、「棚上げ」の構造
    - いきなり全部は無理、だから一歩ずつやってみよう
- 例: n 個の数の和を求める
  - いきなり n 個は無理
    - a_0　a_{k-2} まで足せたとして a_{k-1} をどうするか考える
    - sum(a,k) = sum(a,k-1) + a_{k-1} という再帰的定義の発想
  - 後は n が 0 の場合を考えればよい
    - sum(a,0) = 0
  - 再帰的定義
    - sum(a,k) = if k == 0 then 0 else sum(a,k-1) + a_{k-1}

# 再帰を利用する場合

- 再帰かループか？
  - (個人的には..) ループより再帰の方が自然.. しかし..
  - 純粋な計算でない場合は、ループの方が無難かも
    - 配列の処理はループがよさげ
- 絶対に再帰にしたい場合
  - 再帰的に定義されているものを扱う場合
    - 関数が再帰的な構造を持つ cf. 階乗
  - 再帰的に定義されているデータを扱う場合
    - リスト、木、etc..
    - 再帰的な構造を持つものは多い (結果的に再帰的な関数が..)
- 数学的に物を考える
  - 帰納的に考えた方が、自然 ( なはず... )
    - 再帰的な関数がすらすらかけるようにする
- 再帰的定義のキモ
  - 問題を二つに分ける
    - k までできたら k+1 の時をどうすればよいか？
    - k = 0 の時の結果を考える

# 自然数と再帰

- 自然数の再帰的定義
  - ペアノの公理
    - 0 は 自然数
    - n が自然数なら n' ( n + 1 ) も自然数
    - 自然数は上記の形で得られるもののみ
- 自然数の計算は ' ( +1 ) から帰納法で計算される ( sample-006.c )
  - 足し算
    - n + 0 = n
    - n + m' = (n+m)'
  - かけ算
    - n * 0 = 0
    - n * m' = (n * m) + n
  - 冪乗
    - n^0 = 0
    - n^(m') = (n^m) * n

# 配列の再帰的な定義

- 配列を再帰で処理する
  - 配列要素の再帰的定義
    - a[0] は配列の要素
    - a[0]　a[n] が配列の要素なら a[n+1] も配列の要素
  - 配列の要素を処理するプログラム
    - 配列の再帰的構造を利用する
- 再帰で定義された配列処理関数 ( sample-007.c )
  - 配列の要素の表示
  - 配列の要素の入力
  - 配列の要素の総和を計算する

# 性質から再帰へ

- 最大公約数
  - n と 0 の最大公約数は n
    - gcm( n, 0 ) = n
  - n と m の最大公約数と m と n mod m の最大公約数は同じ
    - gcm( n, m ) = gcm( m, n % m )
  - 最大公約数の性質(等式)から再帰プログラムが作成できる

# 再帰呼び出しと実行順序

- 再帰呼び出しの順序
  - 前に呼ぶ
    - 前から後に実施されるようにみえる
  - 後で呼ぶ
    - 後から前に実施されるようにみえる
  - 途中で呼ぶ
    - 入れ子になる
- 配列の総和の計算
  - 計算の順序に注意
    - 小さい方から足すのと大きい方から足すのでは結果が違う

# 多重再帰

- フィボナッチ数
  - fib(0) = 1, fib(1) = 1, f(n) = f(n-1)+f(n-2)
- ハノイの塔
  - 1　n の番号のついた穴のあいた円板がある
    - 大きさが全て異なり、番号が小さいものが小さい
    - 小さい円板の上に大きい円板を載せてはいけない
  - A,B,C の三つの棒がある
    - 最初は A に、1 から n の円板が大きい順に載っている
    - この円板を C に全て動かしたい
  - 円板の移動規則
    - 円板は、一度に 1 枚しか移動できない
    - 円板は、小さいものの上に大きいものを載せてはいけない

# 課題

- Text p.197 の演習 8-6
  - 

# おわり

終了