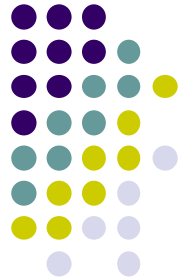
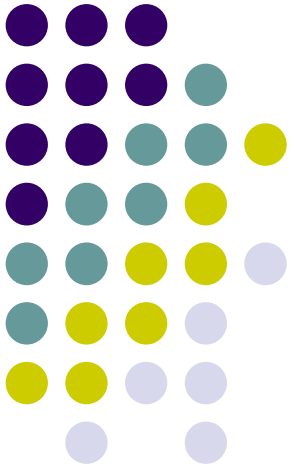


今日の内容

- アルゴリズムとプログラムの設計
  - トップダウンプログラミング
  - サブルーチンコール
  - 再帰呼出し

「復習」トップダウンプログラミング



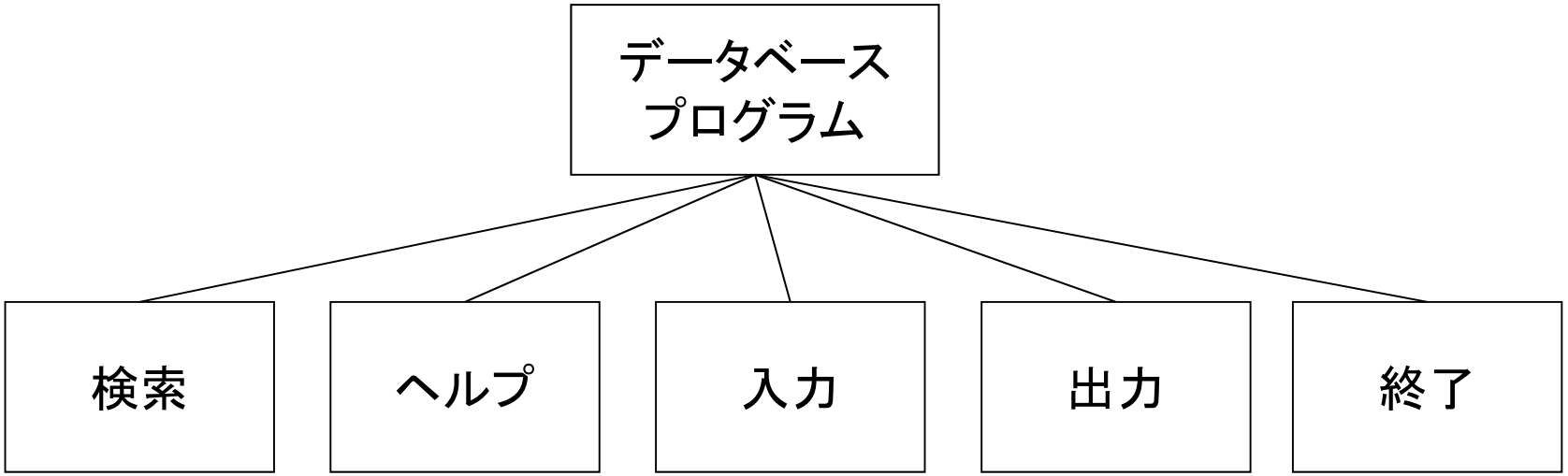
「復習」複雑な問題に対するプログラム

- コンピュータ科学の中心課題は複雑さとその処理
  - 単純な問題は実はコンピュータを使うまでもない
- 高度に複雑な問題をうまく処理する方法
  - 問題を扱いやすくできるよう表現する
  - トップダウンアプローチ/段階的詳細化
    - 問題をより簡単なサブタスクに分解し
    - さらに簡単に理解できるレベルにまで各サブタスクを繰り返し分解する

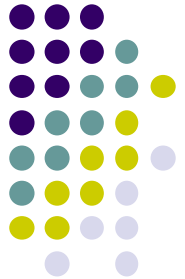
「復習」サブタスクへの分解の例

- 入力したデータを格納しておき
- 格納されたデータの一覧を出力したり
- 問合せに対して適合するデータを検索して出力する

プログラム=データベースプログラム



「復習」サブタスクでの処理



- ヘルプ: ヘルプを表示する
- 終了: プログラムを終了する
- 入力:
  - メモリ内に新しい記憶場所を確保する
  - 入力された新しいデータをそこに格納する
- 出力:
  - データが入っている記憶場所ごとに
  - その内容を出力する

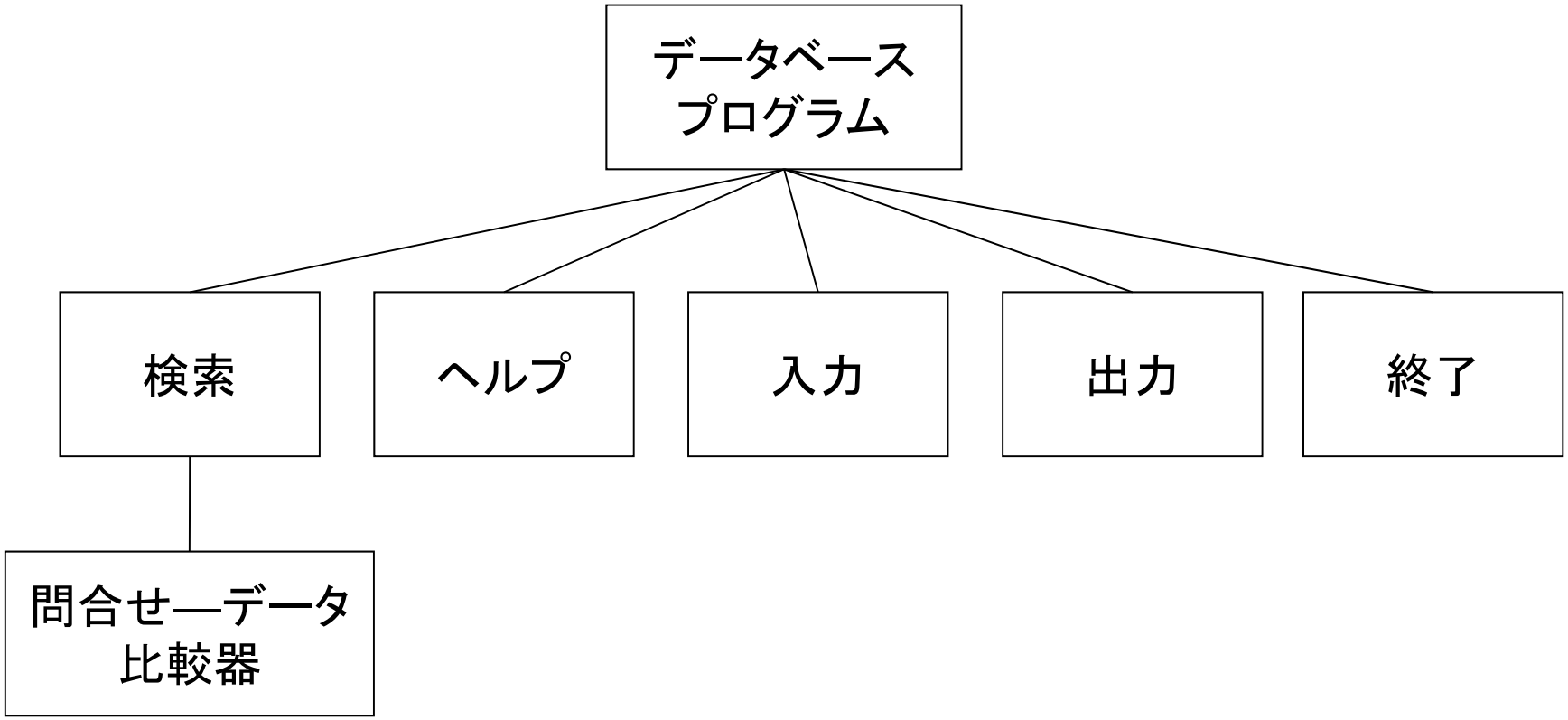
「復習」サブタスクでの処理 (続き)

- 検索:
  - ユーザからの問合せを受付ける
  - データが入っている各記憶領域を検索し
    - 問合わせに適合するデータであれば，それを出力する

↓まだ少し複雑すぎるので．．．

- 検索
  - ユーザからの問合せを受付ける
  - データが入っている各記憶領域を検索し
    - 問合せ—データ比較器を呼び出す
    - 比較機が「match」と報告すれば
      - そのデータを出力する

「復習」さらなるサブタスクへの分解の例



「復習」サブルーチン

- 問題をサブタスクに分解

↓

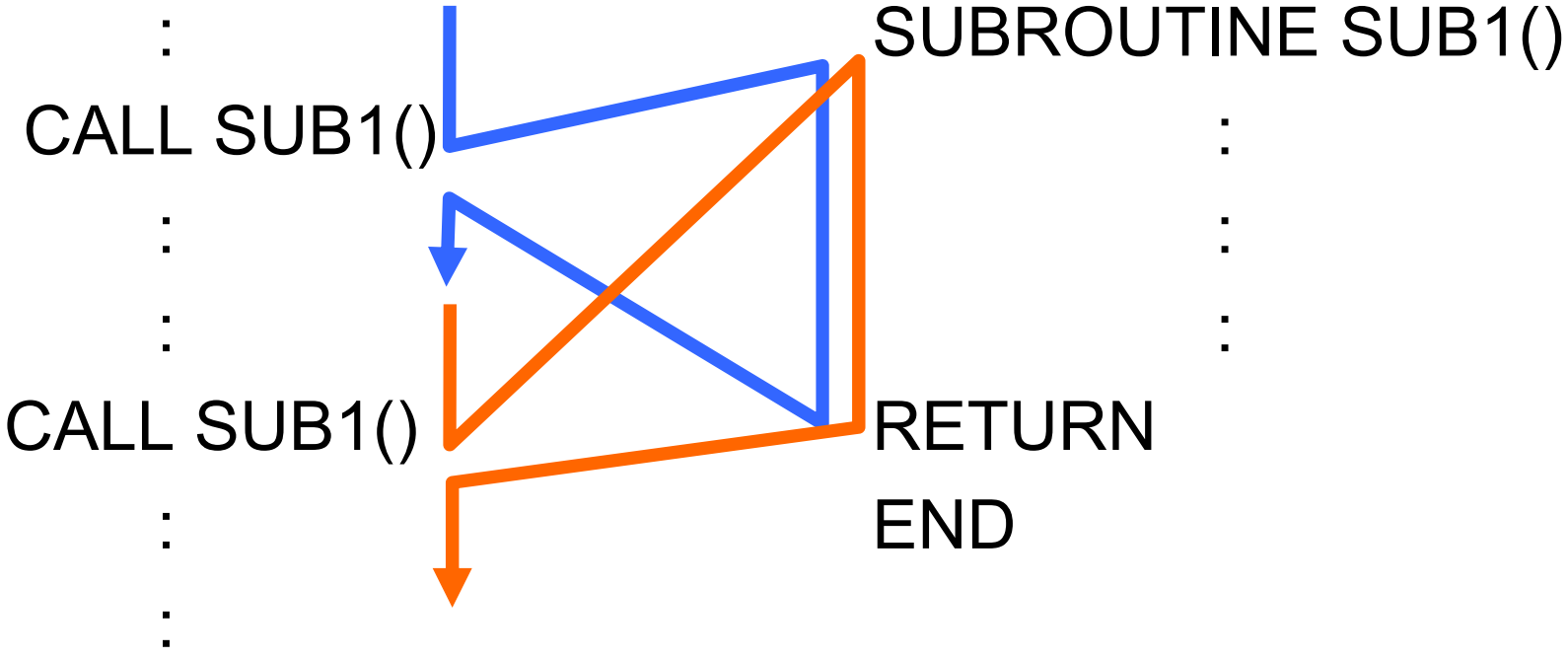
- プログラムもサブタスクごとに記述したい

↓

- サブタスクをひとつのまとまったプログラムとして記述
- 各サブタスクのプログラムを必要に応じて呼び出すプログラムを作ってプログラムを完成させる

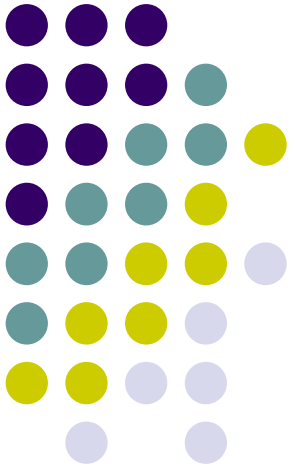
- サブタスクのプログラム: サブルーチン
- サブルーチンを呼び出すプログラム: メインルーチン
- サブルーチンを呼び出すこと: サブルーチン呼出し，サブルーチンコール

「復習」サブルーチンコール



- CALLがあると制御が対応するサブルーチンに移る
- RETURNがあると呼んだCALL文の直後に制御が移る

確認問題10-1

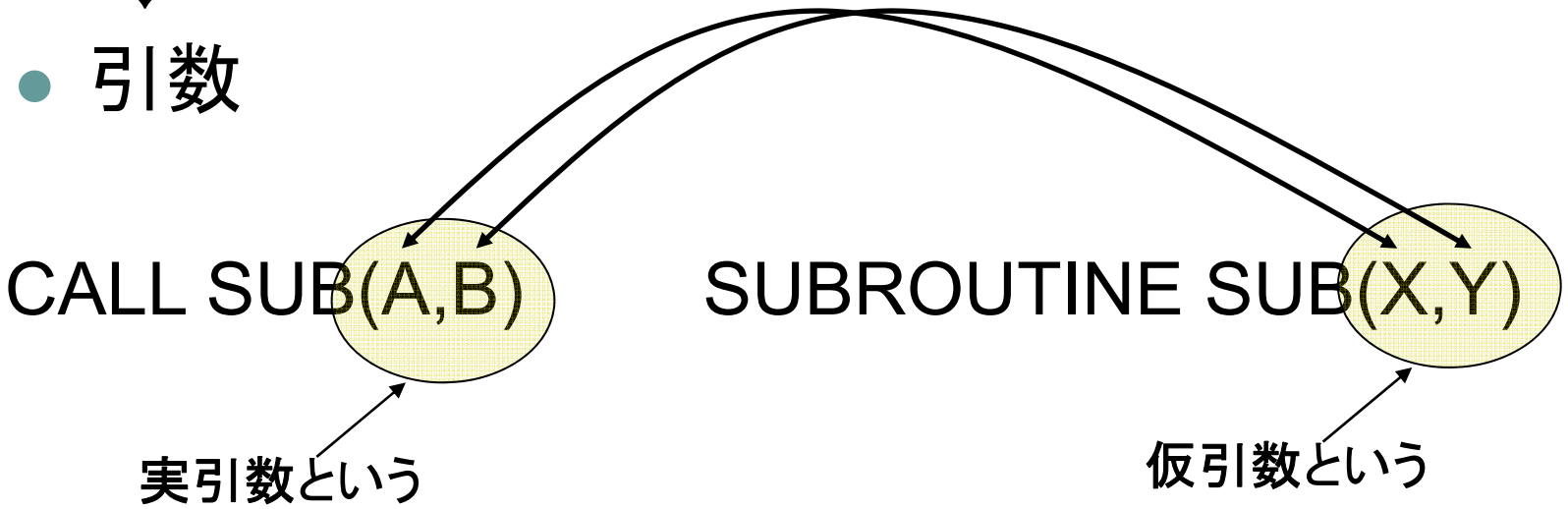


サブルーチンの値を渡したい

- メインルーチンからサブルーチンに値を渡す
  - なんのために?: メインルーチンで得た値に基づいてサブルーチンでの処理を制御する

↓

  - 引数



関数

- サブルーチンでの処理結果をメインルーチンに返したい

  ↓

- **返り値**
- 返り値のあるものを**関数**と呼ぶ

言語	返り値なし	返り値あり
FORTRAN	サブルーチン副プログラム	関数副プログラム
Pascal	手続き	関数
C	(関数)	関数

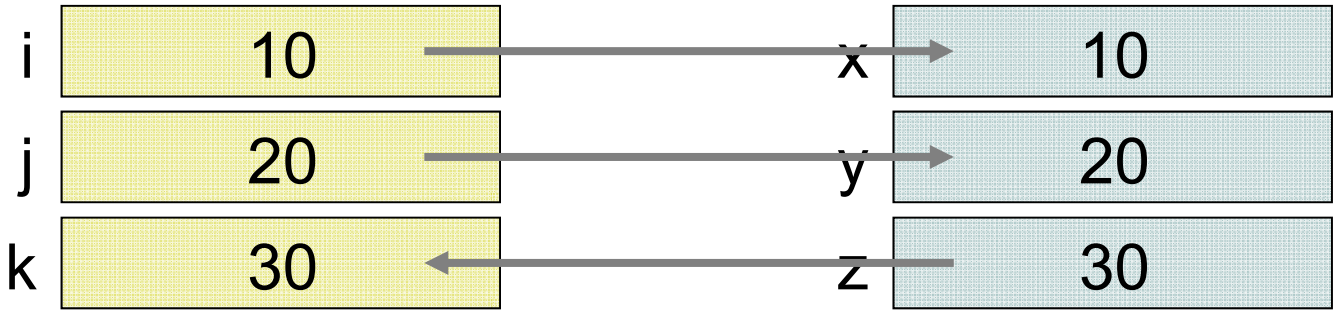
引数の渡し方

- 値渡し (call by value)
  - 実引数の値を仮引数（変数）にコピー
- 参照渡し (call by reference)
  - 実引数のアドレスが仮引数のアドレスとして用いられる
- 名前渡し
  - 仮引数が出現しているところを文面上，実引数（式でも良い）で置き換える

値渡し

```
main () {
    int i, j, k;
    i = 10;
    j = 20;
    k = func1(i, j);
      :
}
```

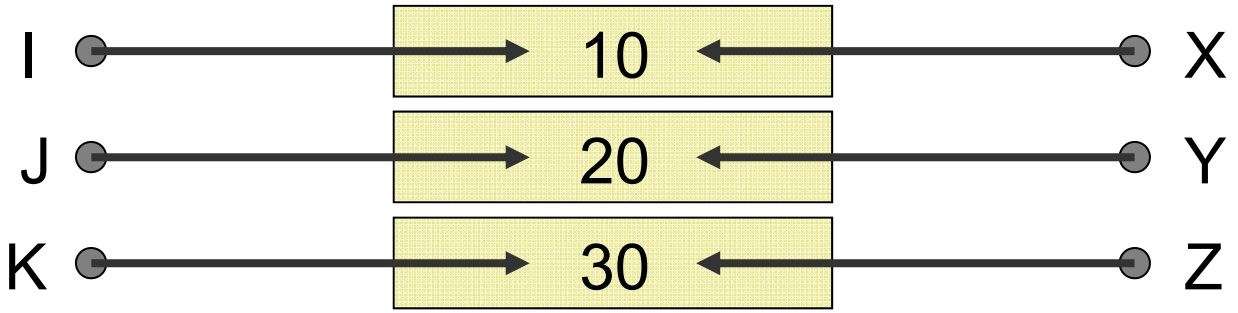
```
int func1(int x, int y) {
    int z;
    z = x + y;
    return z;
}
```



参照渡し

```
INTEGER I, J, K
I = 10
J = 20
CALL SUB1(I, J, K)
      :
END
```

```
SUBROUTINE SUB1(X, Y, Z)
INTEGER X, Y, Z
Z = X + Y
END
```



各言語での引数の渡し方

- FORTRAN: 参照渡し
- Pascal: 参照渡し と 値渡し
  - procedure sub(var a,b: integer) 参照渡し
  - procedure sub(a,b: interger) 値渡し
- C: 値渡し

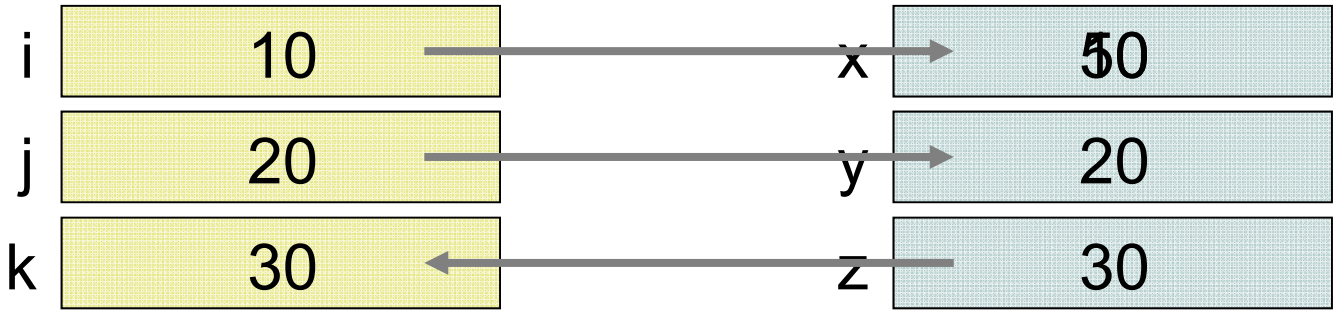
- Cで参照渡しを実現するには？
  - ポインタ型

注: 参照渡しの言語でも実引数が直数値や数式の場合は値渡しとなる

値渡し(例2)

```
main () {
    int i, j, k;
    i = 10;
    j = 20;
    k = func1(i, j);
      :
}
```

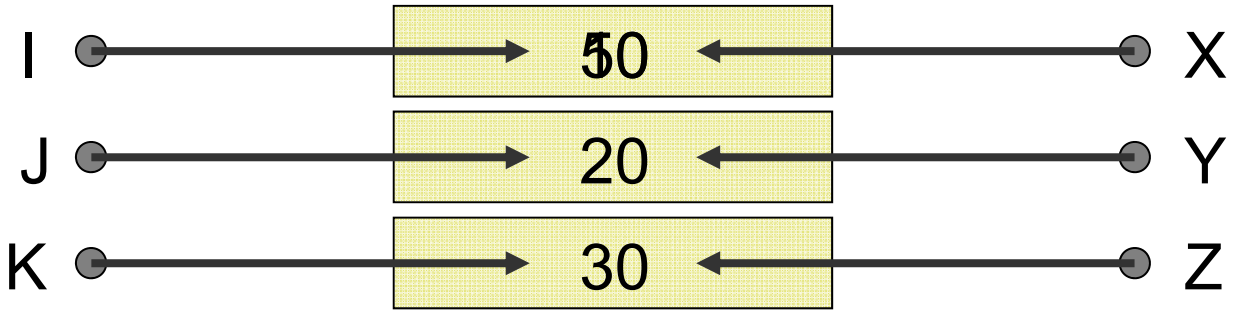
```
int func1(int x, int y) {
    int z;
    z = x + y;
    x = 50;
    return z;
}
```



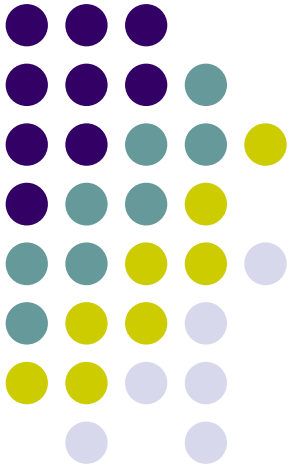
参照渡し(例2)

```
INTEGER I, J, K
I = 10
J = 20
CALL SUB1(I, J, K)
      :
END
```

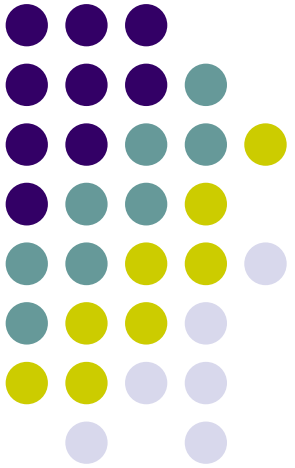
```
SUBROUTINE SUB1(X, Y, Z)
INTEGER X, Y, Z
Z = X + Y
X = 50
END
```



確認問題10-2



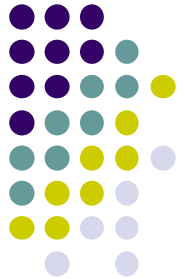
再帰



条件判断，繰返しを使ってどんなプログラムでも書けるのか？

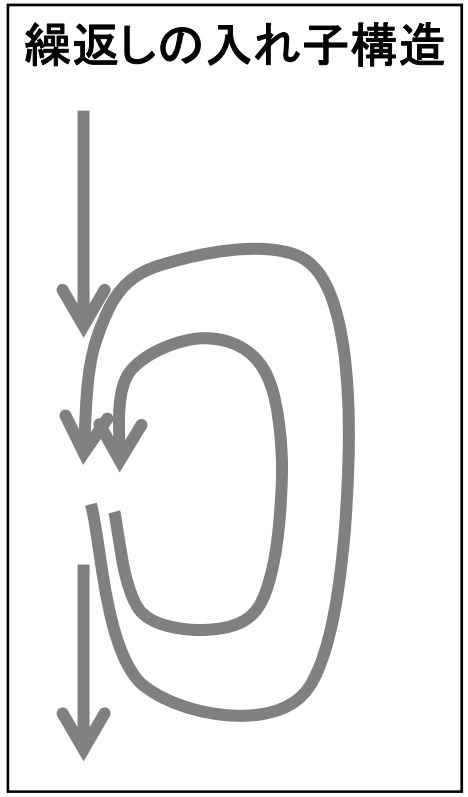
- かなりのものは書ける
  - 繰返しを入れ子にしてやれば，相当複雑な処理もできる
  - サブルーチンや関数に処理まとめてプログラムを理解しやすくできる
- 本当にどんなプログラムも書ける?

【問題意識】順列を求める例題



- a,bのすべての順列を求めよ
  - リスト変数Lに(a,b)を格納
  - 変数 X にLから1つの要素を順次取出して格納しながら以下を繰り返す
    - MにLからXの要素を除いたリストを格納
    - YにMの要素を格納
    - X, Yの順に出力

- L: (a,b)
- X: a
  - M: (b)
  - Y: b
  - 出力: a,b
- X: b
  - M: (a)
  - Y: a
  - 出力: b,a



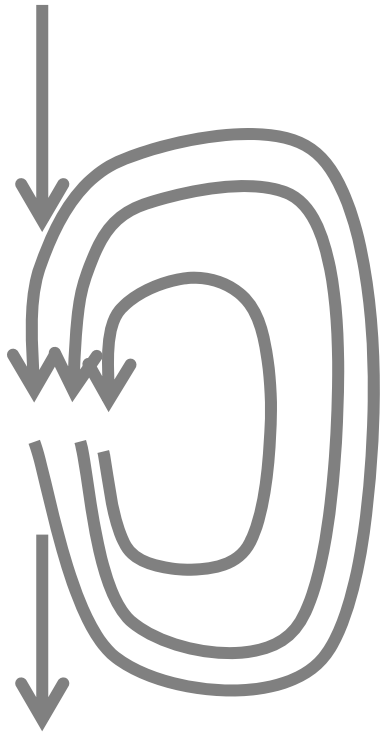
【問題意識】
順列を求める例題 (続き)

- a,b,cのすべての順列を求めよ
  - リスト変数Lに(a,b,c)を格納
  - 変数 X にLから1つの要素を順次取出して格納しながら以下を繰り返す
    - MにLからXの要素を除いたリスト格納
    - 変数YにMから1つの要素を順次取出して格納しながら以下を繰り返す
      - NにLからYの要素を除いたリストを格納
      - ZにNの要素を格納
      - X, Y, Zを出力

【問題意識】
順列を求める例題 (続き)

- L: (a,b,c)
- X: a
  - M: (b,c)
  - Y: b
    - N: (c)
    - Z: c
    - 出力: a,b,c
  - Y: c
    - N: (b)
    - Z: b
    - 出力: a,c,b
- X: b
  - M: (a,c)
  - Y: a
    - N: (c)
    - Z: c
    - 出力: b,a,c
  - Y: c
    - N: (c)
    - Z: a
    - 出力: b,c,a
- X: c
  - M: (a,b)
  - Y: a
    - N: (b)
    - Z: b
    - 出力: c,a,b
  - Y: b
    - N: (a)
    - Z: a
    - 出力: c,b,a

繰返しの3重入れ子構造



【問題意識】
順列を求める例題 (続き)

- a,b,c,dのすべての順列を求めよ
  - リスト変数Lに(a,b,c,d)を格納
  - 変数 X にLから1つの要素を順次取出して格納しながら以下を繰り返す
    - MにLからXの要素を除いたリスト格納
    - 変数YにMから1つの要素を順次取出して格納しながら以下を繰り返す
      - NにLからYの要素を除いたリストを格納
      - ZにNから1つの要素を順次取出して格納しながら以下を繰り返す
        - OにNからZの要素を除いたリストを格納
        - WにOの要素を格納
      - X, Y, Z, Wを出力

【問題意識】
順列を求める例題 (続き)

- a,b,c,d,eのすべての順列を求めよ
  - リスト変数Lに(a,b,c,d,e)を格納
  - 変数 X にLから1つの要素を順次取出して格納しながら以下を繰り返す
    - MにLからXの要素を除いたリスト格納
    - 変数YにMから1つの要素を順次取出して格納しながら以下を繰り返す
      - NにLからYの要素を除いたリストを格納
      - ZにNから1つの要素を順次取出して格納しながら以下を繰り返す
        - OにNからZの要素を除いたリストを格納
        - WにOから1つの要素を順次取出して格納しながら以下を繰り返す
          ・PにOからWの要素を除いたリストを格納
          ・UにPの要素を格納
      - X, Y, Z, W, Uを出力

【問題意識】
順列を求める例題 (続き)

- a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,zのすべての順列を求めよ

!!

- すごく大変そうだけど(26重ループ)，努力と根性でなんとかなるか．．．

そこで問題とプログラムを良く見てみよう

- 要素数が増えると入れ子は確かに深くなっているが
  - ループでやっていることは似ている: 変数名が違うくらいでほぼ同じ
  - 入れ子が深い方に行くにしたがって，問題が簡単に(リストの長さが短く)なってくる
  - 入れ子が一番深い部分はすごく単純
    - リストに要素が1個しかない

そこで問題とプログラムを良く見てみよう (続き)

- すなわち
  - 入れ子になったそれぞれのループでの，処理は似通っている
  - 入れ子が1段深いところでは，問題が1段階簡単になっている
  - 入れ子が一番深いところでは，問題が非常に簡単
  - 入れ子の外側の処理も，一段深い入れ子がうまくやってくれるという前提で，比較的簡単に書いてある

ひらめいた!!

- (複雑そうな)問題の処理を
  - 1段簡単な問題の処理
  - それを用いた今の問題の処理

に分割する

- 一番簡単な問題に対する処理を考える

具体例

- Nが与えられた時に，Nの階乗
  N!=1×2×3×．．．×(N-1)×N
  を求める
- 繰り返しだけでも簡単にできるけど．．．
  - F←1
  - Jを1からNまで繰り返す
    - F←F*J
- 練習ということで，再帰を使ったアルゴリズムを考えてみよう

階乗を再帰を用いて計算するアルゴリズム

- 問題を分ける
  - 1段階簡単な問題: (n-1)!
  - その解を用いて問題を解く処理: n×(n-1)!
- 一番簡単な場合の処理: 0! = 1

階乗を求める関数fact(n)のアルゴリズム

  - もし引数が0なら
    - 1を返り値として帰る
  - そうでなければ
    - n-1を引数として自分自身を呼ぶ
    - 得られた返り値にnをかけた値を返り値として帰る

再帰呼出しの基本

- (複雑そうな)問題の処理を
  - 1段簡単な問題の処理
  - それを用いた今の問題の処理

に分割する

- 一番簡単な問題に対する処理を考える

再帰呼出しの基本(続き)

- 次のようなアルゴリズムによる関数を作る

- もしも，引数として一番簡単な問題が与えられたら
  - 一番簡単な問題に対する処理を行い解を返す
- そうでなければ
  - 一段簡単な問題を引数として自分自身を呼出し，その解を得る
  - 得た解を用いて問題を処理し解を得る

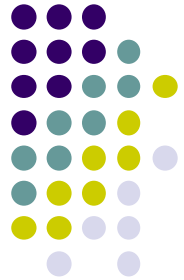
階乗を再帰を用いて計算するプログラム

```
int fact(int n) {
  int r;
  if (n == 0) {
    r = 1;
  } else {
    r = fact(n-1)*n;
  }
  return r;
}
```

動作

- n=3の場合
- fact(3): fact(2)*3を計算しようとする
  - fact(2): fact(1)*2を計算しようとする
    - fact(1): fact(0)*1を計算しようとする
      - fact(0): 1を返り値として帰る
    - 1*1=1なので1を返り値として帰る
  - 1*2=2なので2を返り値として帰る
- 2*3=6なので6が答え

動作 (続き)



- n=4の場合
- fact(4): fact(3)*4を計算しようとする
  - fact(3): fact(2)*3を計算しようとする
    - fact(2): fact(1)*2を計算しようとする
      - fact(1): fact(0)*1を計算しようとする
        - fact(0): 1を返り値として帰る
      - 1*1=1なので1を返り値として帰る
    - 1*2=2なので2を返り値として帰る
  - 2*3=6なので6を返り値として帰る
- 6*4=24なので24が答え

動作 (続き)

- n=5の場合
- fact(5): fact(4)*5を計算しようとする
  - fact(4): fact(3)*4を計算しようとする
    - fact(3): fact(2)*3を計算しようとする
      - fact(2): fact(1)*2を計算しようとする
        - fact(1): fact(0)*1を計算しようとする
          - fact(0): 1を返り値として帰る
        - 1*1=1なので1を返り値として帰る
      - 1*2=2なので2を返り値として帰る
    - 2*3=6なので6を返り値として帰る
  - 6*4=24なので24を返り値として帰る
- 24*5=120なので120が答え

順列を再帰によって求めるアルゴリズム

- 問題を二つにわける
  - 一段簡単な問題: 要素数がひとつ少ない問題
  - 一段簡単な問題の解を用いた処理:
    - リストのどれかひとつに要素 + その要素を除いたリストにおける順列
    - リストの要素それぞれに対して上記を行う→繰り返し
- 一番簡単な問題に対する処理
  - 要素が1つしかない場合
  - その要素を解とする

順列を再帰によって求めるアルゴリズム (続き)

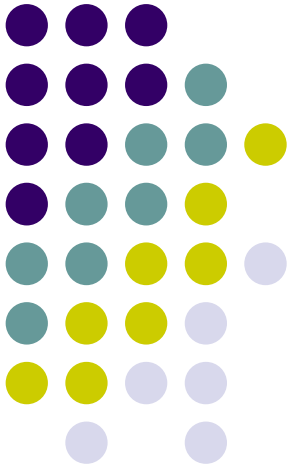
順列を求める関数permのアルゴリズム

- 引数1:順列生成対象のリスト S
- 引数2:その関数が呼ばれる外側ですでにできあがっている部分順列(出力用) T

順列を再帰によって求めるアルゴリズム (続き)

- もしSの長さが1ならば
  - T と S を連結したものを出力
- そうでなければ
  - Sから要素をひとつずつ順次取出しながら(この要素を変数Aに入れる)以下を繰り返す
    - SからAを除いたリストをUとする
    - TにAを連結したリストをVとする
    - Uを引数1，Vを引数2としてpermを呼ぶ

確認問題10-3



今日はここまで
次回の予告

プログラミング – 再帰(続き)