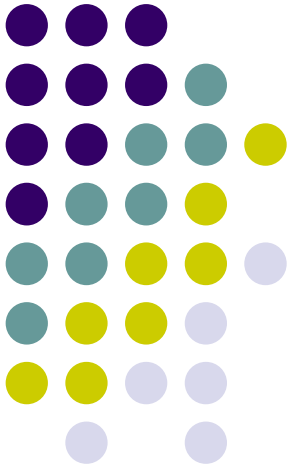
情報処理概論
後半第1回

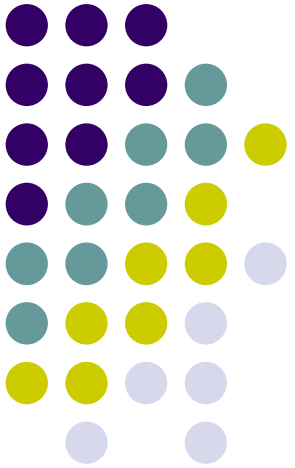
岩橋政宏



今日の内容

- アルゴリズムとプログラム
- 繰り返し
- 誤差
- プログラミングモデルとプログラミング言語

アルゴリズムとプログラム



ソフトウェア設計とプログラム

- ソフトウェアの設計
  - どのように動けば良いか，何ができれば良いか
    → 要求定義
  - 上記の要求を実現するためには，どのような機能(の集合)が必要で，それらがどう連携すれば良いか
    → 基本設計 or モジュール設計

何ができれば良いか

  - 個々のモジュールを，コンピュータのプログラムとして，どうやって実現するか
    → 詳細設計 or プログラム設計
- プログラムの実装

どうやれば良いか

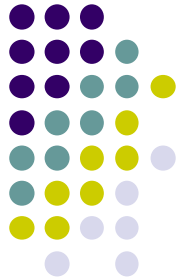
アルゴリズムとプログラム

- アルゴリズム
  - 仕事を実行するための方法または手順の処方箋
  - 一般に入力があり，出力がある
  - 出力の求め方を示す一連の手順 --- 一定のメカニズムの下に動く明確に定義された一連の動作 --- を含んでいなければならない
  - 誰が，または誰が，その手順を実行するかは指定しない
    - 実行者は機械でも良いし，人間でも良い
    - 自然語で書いても良いし，人工言語で書いても良い

アルゴリズムとプログラム (続き)

- プログラム
  - アルゴリズムをコンピュータで実行させるべく，コンピュータ向けの言語で記述(コード化)したもの
- アルゴリズム作成とコード化は別のもの
  - アルゴリズム作成: 仕事を実行するための方法の発見
  - コード化: 発見した方法のコンピュータ言語への翻訳

プログラムとは何か



- プログラム＝
  - コンピュータにある仕事をさせるための
  - 命令の並び
- 命令＝
  - メモリ上のデータを操作する
  - 入出力機器にデータを送って制御する

メモリ上のデータの操作

- メモリのあるアドレスからデータを読み出す
  → プロセッサ内の一時的なデータ格納場所=レジスタに格納
- メモリの別のアドレスからデータを読み出す
  → 別のレジスタに格納
- レジスタに格納された値同士で計算
  - 加算，減算，乗算，除算
  - AND, OR, XOR
  - シフト，ローテート

  を行い，結果をレジスタに格納する
- 結果のレジスタ値をメモリのあるアドレスに格納する

入出力の制御

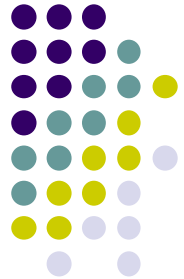
- 出力ポートのあるアドレスに，周辺機器を制御する命令に対応するデータを出力

  → 周辺機器が命令にしたがって動作

- 入力ポートのあるアドレスからデータを読み出す

  → 周辺機器の状態や保持しているデータを取込み

再びプログラム



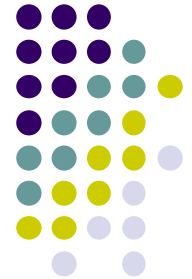
- メモリー上のデータの操作
- 入出力装置の制御

を順番に実行するという動作でかなりいろいろなことができる

しかし．．．例えばこんなことはできるだろうか?

- 総和の計算
  - 1から10までの総和を計算する
  - 1から100までの総和を計算する
  - 1からNまでの総和を計算する
- 外部の状況を判断して動作するロボット
  - 壁にぶつかりそうになったら曲がる
  - 進路上に障害物があったらよける

プログラムのためにさらに必要な要素



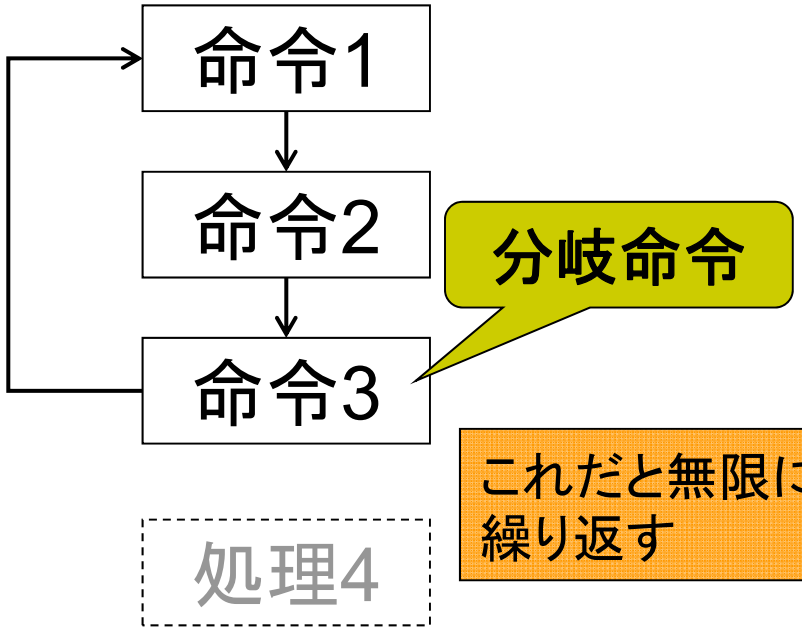
- 判断
  - 2つのレジスタの値同士が等しいか・大きいか・小さいか判別
  - レジスタの値が指定した値に等しいか・大きいか・小さいか判別
- 分岐
  - 次の番地の命令を実行するという通常の動作を変えて，指定した番地の命令から実行する(無条件ジャンプ)
  - 上記の判断に基づき，特定の条件が成り立つ場合は指定した番地から実行し，そうでなければ次の番地から実行する(条件付きジャンプ)

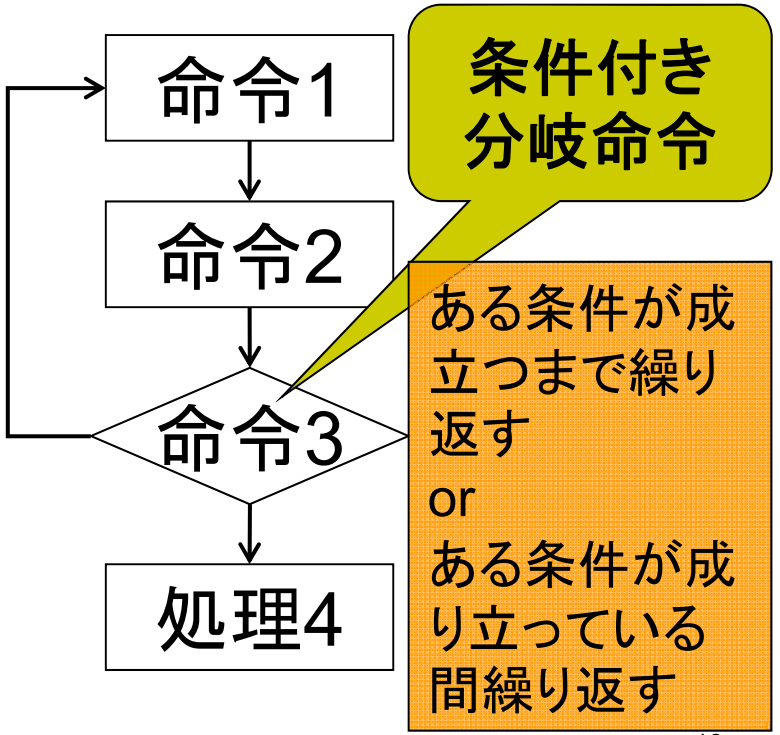
判断と分岐が加わると．．．

- 条件に基づいて状態や行動を決定するプログラム
  - 故障の診断
  - 携帯電話の料金プランをどれにするか?
- 条件に基づいて動作が変るプログラム
  - ロボット

繰り返し：判断と分岐の組み合わせ

- 計算や動作をある条件に達するまで**繰り返す**
  - 例：1～Nまでの総和





繰り返し (続き)

- 同じ計算式を用いて，すこしずつ変数の値を変えて計算した結果を求める
  - 数種類の値に対する計算ならば，計算式のコードを羅列すればできる
    - 1.0から5.0までの実数値(1.0きざみ)に対して，その2倍の値を計算し出力するプログラム

```
x = 1.0; d = x * 2;  printf(("%6.2f,%6.2f¥n", x, d);
x = 2.0; d = x * 2;  printf(("%6.2f,%6.2f¥n", x, d);
x = 3.0; d = x * 2;  printf(("%6.2f,%6.2f¥n", x, d);
x = 4.0; d = x * 2;  printf(("%6.2f,%6.2f¥n", x, d);
x = 5.0; d = x * 2;  printf(("%6.2f,%6.2f¥n", x, d);
```

繰り返し (続き)

- 非常に多くの様々な変数値に対して計算したい場合はどうする?
  - 三角関数の数表を0.01度きざみで作る
- プログラム動作中に計算したい変数値の範囲や計算終了の条件が決まる場合はどうする?
  - プログラムが動き出しから，変数値の範囲をキーボードで入力する
  - 直前の計算結果と今の計算結果との差が0.001以下になったら計算終了

繰り返し (続き)

- 例題
  - 1.0から10.0までの実数値(1.0きざみ)に対し
  - その2倍の値を計算し
  - 元の値と計算された値を並べて出力するプログラム

繰り返し (続き)

- Pascal

```
program Double;
var
    d, x: real;
begin
x := 1.0;
while x <= 10.0 do
 begin
 d := 2.0 * x;
 writeln(x:6:2,d:6:2);
 x := x + 1.0;
 end;
end.
```

- C

```
main() {
 float d, x;
 x = 1.0;
 while(x <= 10.0) {
  d = 2.0 * x;
  printf("%6.2f,%6.2f¥n", x, d);
  x += 1.0;
 }
}
```

繰り返しにおける動作

命令		x	d
x =1.0	1.0		
(test) x <= 10.0	(true)	1.0	
d = 2.0 * x		1.0	2.0
printf		1.0	2.0
x = x + 1.0		2.0	2.0
(test) x <= 10.0	(true)	2.0	2.0
d = 2.0 * x		2.0	4.0
printf		2.0	4.0

繰り返しにおける動作 (続き)

命令		x	d
:		:	:
x = x + 1.0		10.0	18.0
(test) x <= 10.0	(true)	10.0	18.0
d = 2.0 * x		10.0	20.0
printf		10.0	20.0
x = x + 1.0		11.0	20.0
(test) x <= 10.0	(false)	11.0	20.0
(終了)			

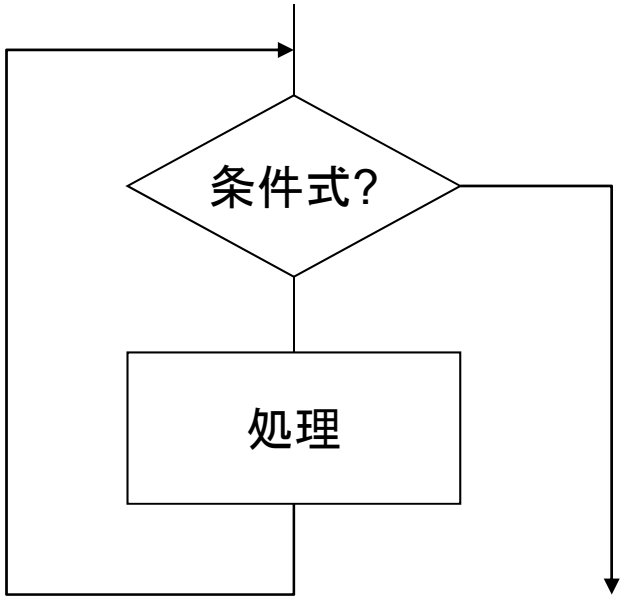
様々な繰り返し文

- 前置判定
  - 条件判定をループの入口で行う
  - Pascal

    while 条件式 do
      処理;

    C

    while(条件式) {
     処理
    }



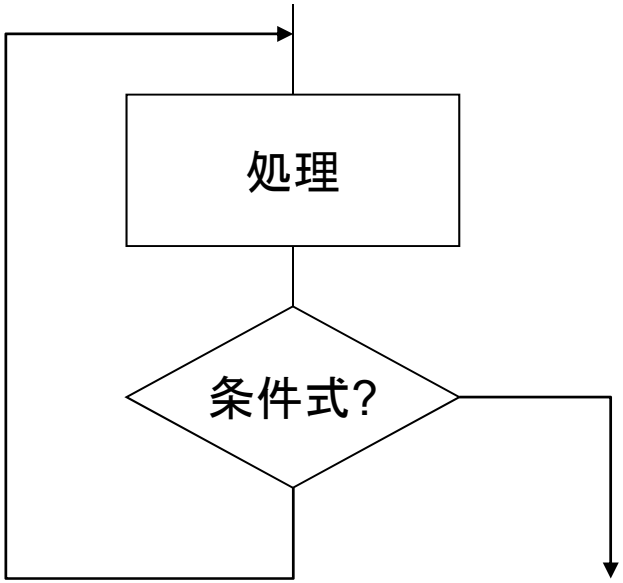
  - 条件式が偽となる場合には処理は一度たりとも実行されない点に注意

様々な繰り返し文 (続き)

- 後置判定
  - 条件判定をループの後部で行う
  - Pascal

    repeat
    　処理
    until 条件式;
  - C

    do {
    処理
    } while(条件式)
  - 条件式が偽でも処理は一度は実行される
  - PascalとCでは繰り返し継続の条件が逆な点に注意
    - Pascal: 条件式が真になるまで(偽の間)繰り返す
    - C: 条件式が真の間繰り返す

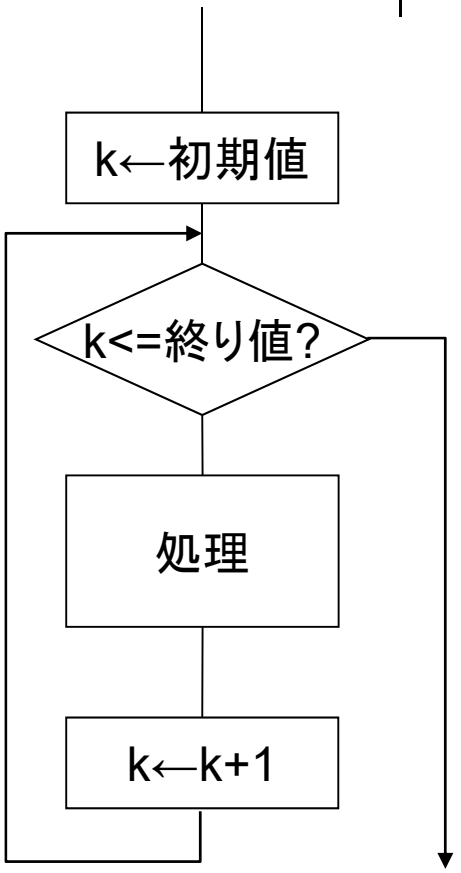


様々な繰り返し文 (続き)

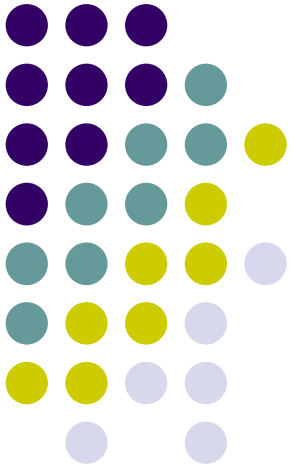
- 制御変数を増加させる繰り返し
  - Pascal

    for k := 初期値 to 終り値 do
      処理;
  - C

    for(k=初期値; k<=終り値; k++) {
     処理
    }
  - kを初期値より1づつ増加させ終り値になるまで繰り返す
  - Cは，より汎用的
    - 繰り返し継続条件が自由に書ける
    - 繰り返す際に実行される命令を自由に書ける



プログラミングモデル とプログラミング言語



コンピュータが直接実行できる命令の形式は?

× 人間が書いたり話したりする言葉(自然言語)

× 厳密に文法が定義され，人間が容易に読むこともできる人工的な言葉

○プロセッサの電子回路が動作しやすいように設計された0と1の羅列からなる命令語(機械語)

ではプログラムとは機械語の命令で書くものなのか

- 初期にはそのとおり

↓

- あまりにも人間的でない
  - プログラム作成に時間がかかる
  - 間違いも多い
  - 精神的に疲れる
  - 他人が書いたものはほとんど理解できない

↓

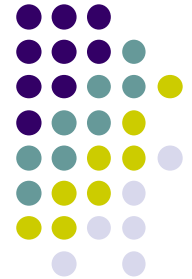
- もう少し人間的な書き方でプログラムをしたい!!

ではプログラムとは機械語の命令で書くものなのか(続き)

- アセンブリ言語
  - 機械語と1対1対応するが，0と1の羅列ではなく，人間にとって意味がわかる単語を割り当てる
  - 単語を0と1の羅列に変換するためのプログラム→アセンブラ
- 高級言語
  - アセンブリ言語よりもっと人間にわかりやすいプログラムをすることを目指して作られた言語
  - とはいえ，コンピュータのプログラムのためなので．．．
    - 簡潔な文法(構文の解析がしやすい)
    - あいまい性がない

    →普段人間が使っている言葉との隔たりは依然大きい
  - 現代ではプログラミング言語といえば高級言語を指す

プログラミング言語の分類
ーコンパイラ言語とインタープリタ言語ー

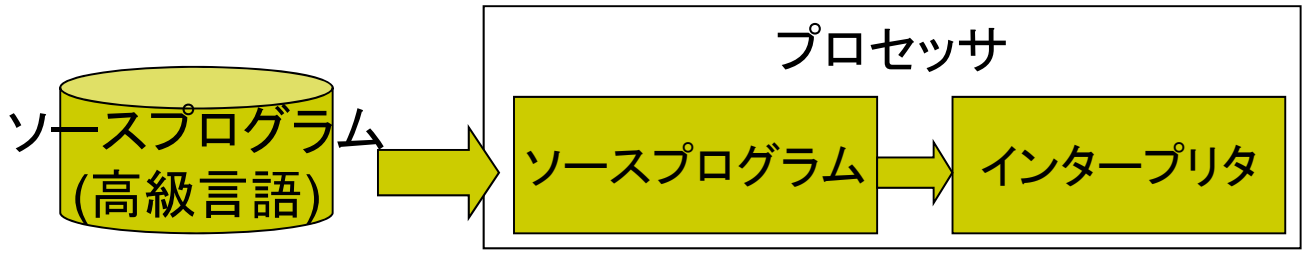


- コンパイラ言語
  - コンパイラというプログラムによって，書かれたプログラムのすべてをあらかじめ機械語に変換
  - コンパイラは変換の際に動作，プログラム実行時は機械語変換されたプログラムが実行される
- インタプリタ言語
  - インタプリタというプログラムが，書かれたプログラムを1行ごとに解釈して実行する
  - プログラム実行時にインタプリタが動作．
  - インタプリタが書かれたプログラムを解釈・実行

復習: コンパイラとインタープリタ (第4回ですでに学習)

- コンパイラ
  - 高級言語で書かれたプログラムを機械語のプログラムに翻訳するプログラム

- インタープリタ
  - 高級言語で書かれたプログラムを解釈実行するプログラム



プログラミング言語の分類
ープログラミングモデルによるー

- 古典的プログラミング言語
  - FORTRAN，COBOL
  - 分岐の形態は機械語とほぼ同じ
    - 動作が複雑なプログラムはジャンプが多くなり飛び先の関係も入り組んでくる←スパゲッティ・プログラム
    - スパゲッティプログラムはプログラムの動作を追うのが大変→デバッグや改造が困難

そこで↓

- 構造化プログラミング言語
  - ジャンプ命令を直接書かない
  - 分岐を含む構造全体を表す命令語を持たせる

プログラミング言語の分類
ープログラミングモデルによるー(続き)

- 構造化プログラミング言語
  - ジャンプ命令を直接書かない
  - 分岐を含む構造全体を表す命令語を持たせる
    - 繰り返し: while，repeat，for
    - 条件判断: if～else～，switch～case～

プログラミング言語の分類
ープログラミングモデルによるー(続き)

- 構造化プログラミング言語
  - 繰り返し
    ```
    while(繰り返し条件) {
     繰り返す処理
    }
    ```
  - 条件判断
    ```
    if (条件) {
     真の場合の処理
    } else {
     偽の場合の処理
    }
    ```
- 古典的プログラミング言語
  - 繰り返し
    ```
     5  IF (繰り返し条件) GOTO 10
        GOTO 20
    10 繰り返す処理
              :
        GOTO 5
    20 ….
    ```
  - 条件判断
    ```
        IF (条件) GOTO 30
        偽の場合の処理
        GOTO 40
    30 真の場合の処理
              :
    40 …
    ```

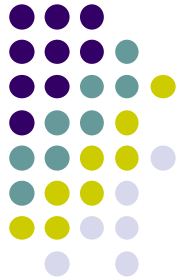
プログラミング言語の分類
ープログラミングモデルによるー(続き)

- 再利用性を高めたい
- 世の中の物事を直接的にプログラム化したい

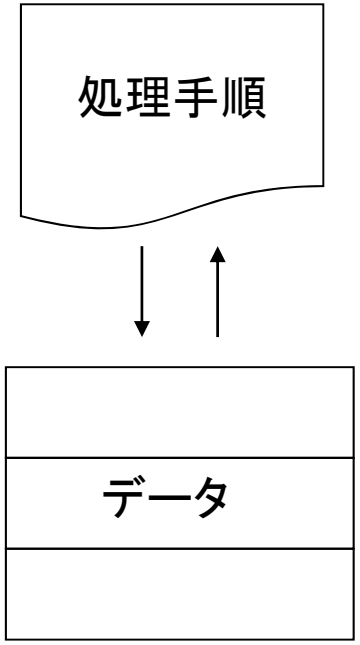
↓

- オブジェクト指向プログラミング言語
  - 従来のプログラミング言語
    - プログラム=データ+命令の制御構造
  - オブジェクト指向プログラミング言語
    - プログラム=オブジェクト+オブジェクト間のメッセージ交換

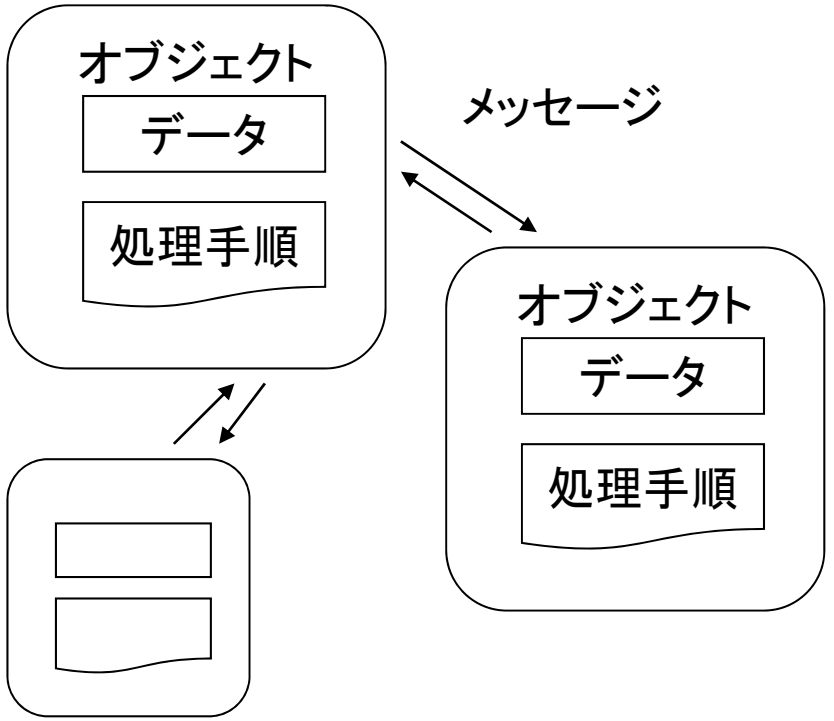
プログラミング言語の分類
ープログラミングモデルによるー(続き)



- 従来のプログラミング言語



- オブジェクト指向プログラミング言語



文字を(あまり)使わないプログラミング言語

- プログラミング言語
  1. コンピュータへの命令の羅列
  2. 人間が考えたアルゴリズムをコンピュータに指示する手段
  3. 人間がやりたい事をコンピュータに伝える手段
- コンパイラは，2→1の変換をしているともとらえることができる
- 2は，必ずしも文字で書かれている必要はない

ロボットを制御するプログラムを考える

ロボットのプログラム (1)

- 動作 = モータの制御
  - モータを○○%の出力でまわす
  - モータを△△回転まわす
- 右，左の車輪にそれぞれ1つずつ
  - 左右を同じ速度でまわす：直進
  - 右か左のどちらかだけをまわす：旋回
  - 一方をゆっくり目にまわす：ゆるい旋回
  - 右と左を逆にまわす：超信地旋回

ロボットのプログラム (2)

- 条件を調べるには = センサ
  - タッチセンサ: ボタンが押されるとONになる
  - 音響センサ: 音が入力されるとONになる
  - 光センサ: 光を当てて，反射光の色や強さに応じた値が出力される
  - 超音波センサ: 超音波により距離を計測し，その距離に応じた値が出力される

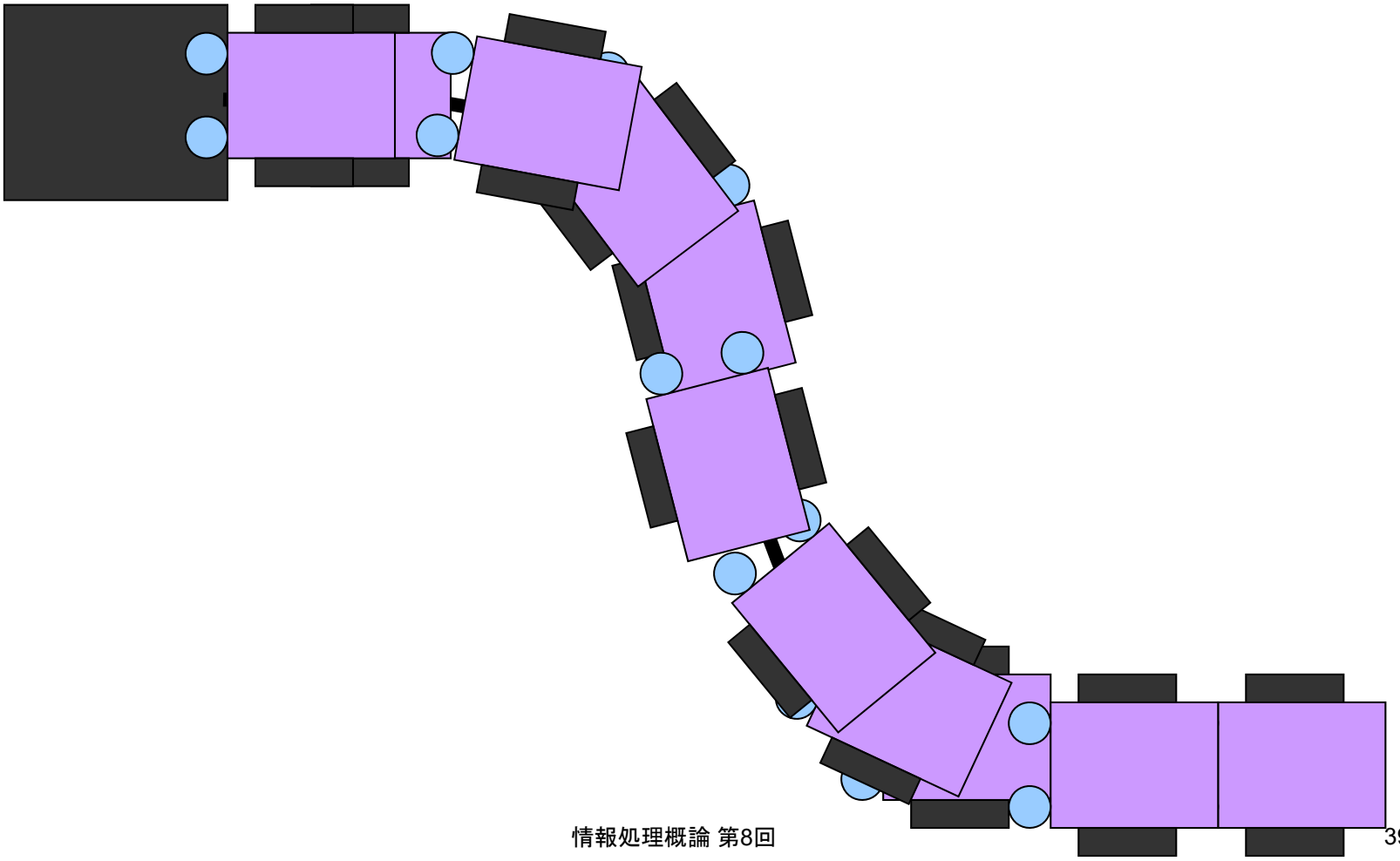
ロボットのプログラム (3)

- センサの出力 (ON/OFFや値) を条件として
  - もしも，〇〇センサが△△ならば，□□をする
  - ××センサが■■になるまで，◎◎を繰返す

といった動作を組合せて

- 思った通りに動作をさせる

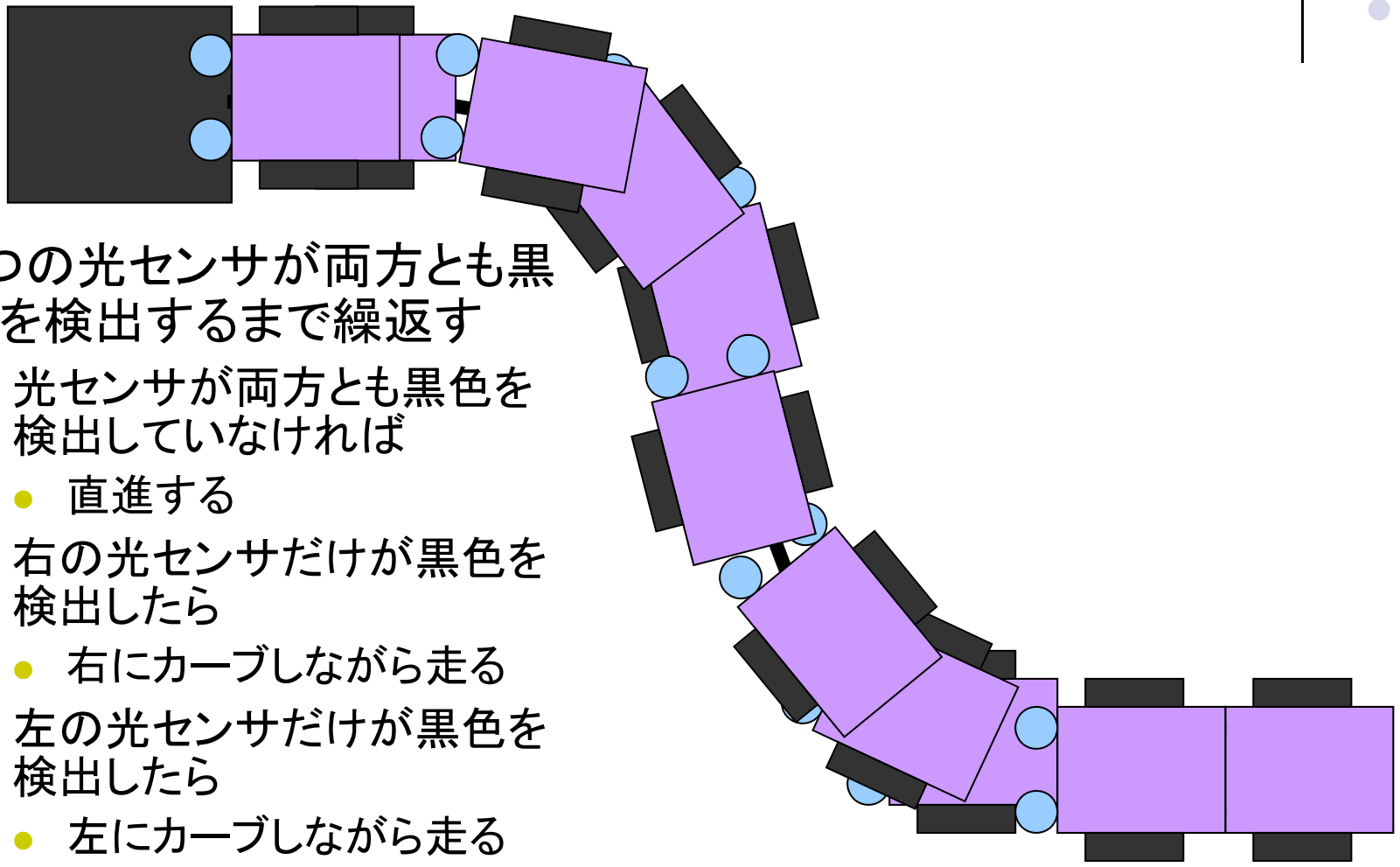
線に沿って走るロボット (ライントレーサ)



ラインレーサのアルゴリズム

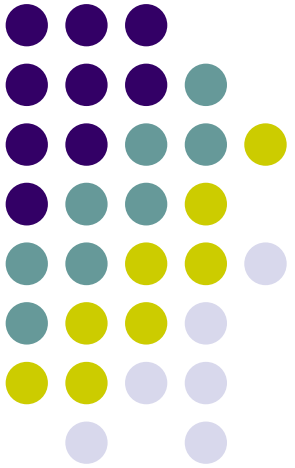
- 2つの光センサが両方とも黒色を検出するまで繰返す
  - 光センサが両方とも黒色を検出していなければ
    - 直進する
  - 右の光センサだけが黒色を検出したら
    - 右にカーブしながら走る
  - 左の光センサだけが黒色を検出したら
    - 左にカーブしながら走る

ライントレーサの動作



- 2つの光センサが両方とも黒色を検出するまで繰返す
  - 光センサが両方とも黒色を検出していなければ
    - 直進する
  - 右の光センサだけが黒色を検出したら
    - 右にカーブしながら走る
  - 左の光センサだけが黒色を検出したら
    - 左にカーブしながら走る

中間試験の解答と解説



今日はここまで