第五回 プログラミング

2002年10月3日

機械語とアセンブリ言語

- CPUが理解できる命令セットは2進法の数値
- 命令セットを暗記用の記号(ニーモニック)で記述する
- ニーモニックで記述されたプログラムをアセンブリプログラムという
- アセンブリ言語で書かれたファイルを「アセンブラ」で処理すると、機械語の実行可能プログラムになる

機械語プログラム

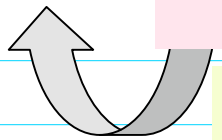
```
アドレス  機械語
0100
0100    000F
0102    0000
0104

0200
0200    B8 0000
0203    B9 0001
0206    03 C1
0208    41
0209    3B 0E 0100
020D    75 F7
020F    A3 0102
0212    C3
0213
```

アセンブリプログラム

```
ラベル     命令語   オペランド
DATA       SEGMENT
N          DW       15
S          DW       0
DATA       ENDS

CODE       SEGMENT
START:     MOV      AX,0
           MOV      CX,1
LABEL1:    ADD      AX,CX
           INC      CX
           CMP      CX,N
           JNZ      LABEL1
           MOV      S,AX
           RET
CODE       ENDS
           END      START
```



アセンブラ

アセンブリプログラムの例

機械語プログラム

```
アドレス  機械語
0100
0100   000F
0102   0000
0104

0200
0200   B8 0000
0203   B9 0001
0206   03 C1
0208   41
0209   3B 0E 0100
020D   75 F7
020F   A3 0102
0212   C3
0213
```

アセンブリプログラム

```
ラベル    命令語    オペランド
DATA      SEGMENT
N         DW        15
S         DW        0
DATA      ENDS

CODE      SEGMENT
START:    MOV       AX,0
          MOV       CX,1
LABEL1:   ADD       AX,CX
          INC       CX
          CMP       CX,N
          JNZ       LABEL1
          MOV       S,AX
          RET
CODE      ENDS
          END       START
```

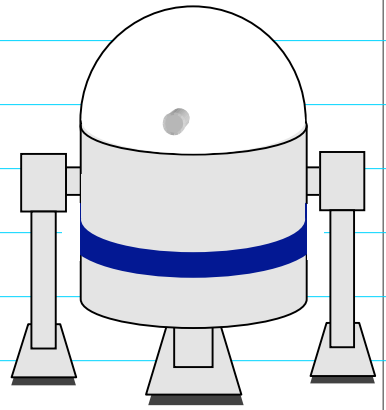
ここからデータセグメント
Nの初期値を15
Sの初期値を0
データセグメントここまで

ここからコードセグメント
AXレジスタに0を代入
CXレジスタに1を代入
(ラベル1) AXにCXを加える
CXを1だけ増加させる
CXをNと比較する(CX-N)
non-zeroならラベル1へジャンプ
SにAXレジスタの値を代入
終了
コードセグメントここまで

1からNまでの数の和を計算してSに記憶する

アセンブリ言語の特徴

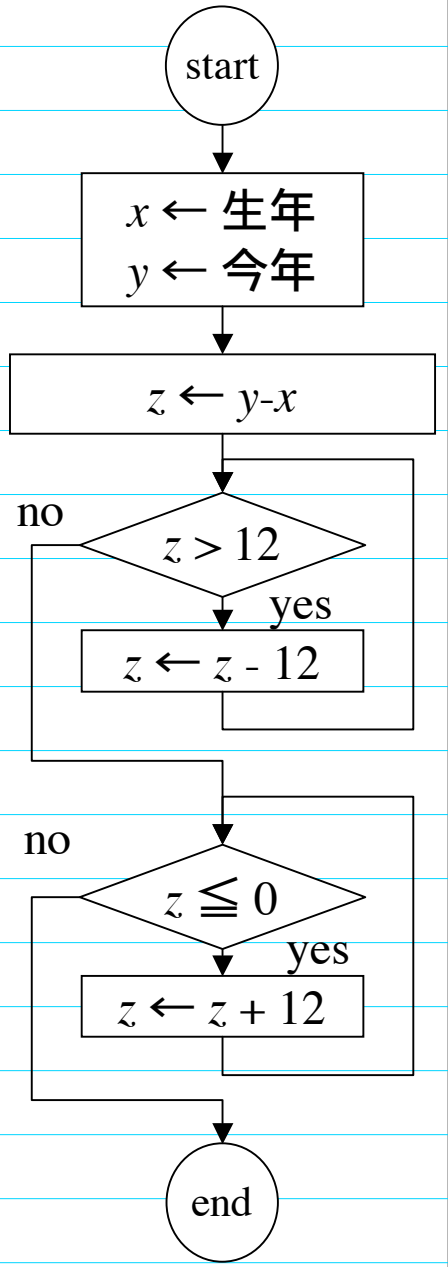
- アセンブリ言語は機械語を人間が読みやすいように翻訳したもの
- CPUの種類毎に命令セットが異なるのでアセンブリ言語も異なる → 再利用や移植が難しい
- 簡単な処理であれば無駄のない効率的なプログラムを作ることができるが、複雑な処理を行う大規模プログラムを作るのは困難



アルゴリズム

- アルゴリズムとは
  「ある目的を実現するための処理の手続き」
- 例：「今年の干支は何？」という質問に答えるにはどのような計算をするか。
  1. 12の干支にそれぞれ1から12までの番号をつける。
  2. 自分の生年(あるいは干支がわかっている年)を x、「今年」を y とする。
  3. $y - x$ を計算する。→ z
  4. z が12より大きければ、12を引く。それ以外は 6.に進む。
  5. 4.に戻る。
  6. z が 0 以下であれば 12を加える。それ以外は 8.に進む。
  7. 6.に戻る。
  8. xの干支に z を加えたものが今年の干支。

フローチャート



数値計算法のアルゴリズム

- 情報科学の重要な基礎分野の一つ
- 最大公約数（ユークリッド法）
- 代数方程式の数値解法
  - 二分法、ニュートン法など
- 行列計算
- 数値積分
- モンテカルロ法

など

さまざまなアルゴリズムのライブラリが提供されている

Fortran数値計算ライブラリ NetNUMPAC

http://netnumpac.fuis.fukui-u.ac.jp/numpac/

Numerical Recipe in C

http://www.ulib.org/webRoot/Books/Numerical_Recipes/bookcpdf.html

など

非数値計算のアルゴリズム

- 計算機の利用範囲の拡大 → 数値計算以外の分野での利用が主体
- 文字列や記号処理など
- 整列化（ソーティング）：
  ランダムに並んだ数字を昇順や降順に並べ変える
  - バブルソート、マージソートなど
- 辞書機能：
  名前や単語の表(辞書)に対して、名前や単語を検索する
  - 総当り検索
  - 2分探索法(バイナリ・サーチ)
  - ハッシュ法 - 名前などの内部表現から配列内の位置情報を決定する

  平均検索回数：総当り検索 ～ $N/2$, 2分探索法 ～ $\log N$

高級言語

- プログラムを作るための言語をプログラミング言語という。
- 人間の思考により近付いたプログラミングを可能にする「高級言語」が開発されてきた
- 「手続き型言語」
  - FORTRAN 科学技術計算 (1956)
  - COBOL　事務処理 (1964)
  - ALGOL　教育・研究用 (1962)
  - PASCAL 教育・構造化(1970)
- 「関数型言語」- LISP 人工知能 (1962)
- 「論理型言語」- Prolog エキスパートシステム (1970)
- 「オブジェクト指向言語」 - SmallTalk (1972)

プログラム言語の変遷

- それぞれの言語の系譜を組むオブジェクト指向言語が現在の主流
  - FORTRAN → FORTRAN90 → ??
  - FORTRAN → BASIC → Visual Basic (1995)
  - ALGOL → C言語(1972) → C++ (1983), Java (1995)
- C は手続き型言語だが、UNIX, Linuxで幅広く普及
- 事務処理ではCOBOLが、一部の科学技術計算ではFORTRANがいまでも根強く使われ続けている
- 過去に作られたソフトウェア資産(=経験,実績,データ,アイデア)をいかに再利用するかが課題
  “Legacy software”　“evolution vs revolution”
- 新しい言語が開発されてから実際に広く普及するまでには数年から10年近くかかる

プログラム文法の概念

- 手続き型言語をもとにした言語には文法上の共通する基本概念がある
- 変数、定数、変数の型宣言
- 配列
- 構造体
- 条件判断・分岐
- 繰り返し
- ポインタ

手続き型言語

- プログラムの実行順を重視しつつ、アセンブリ言語よりもさらに人間が読みやすく、またよく使われる機能を文法に組み込んだもの

FORTRANの例：

```
出力    WRITE(6,100)
100     FORMAT(' ENTER N')
入力    READ(5,110) N
110     FORMAT(I3)
        S=0
計算    DO 120 I=1,N
          S=S+1
120     CONTINUE
出力    WRITE(6,130) S
130     FORMAT(' SUM=',I4)
        STOP
        END
```

C言語の例：

```
#include <stdio.h>

main() {
  int n, i, s = 0;

  printf("Enter n:");
  scanf("%d",&n);
  for( i=1; i<=n; i++) {
    s += i;
  }
  printf("SUM = %d¥n", s);
}
```

関数型言語

- プログラムを「関数の集まりの定義」と捉える
- 自然言語処理、自動翻訳システム、知識データベースなどの人工知能の研究に応用されている
- 関数の中で自分自身を呼ぶ「再帰呼び出し」を使うことでループ処理が可能

LISPの例：

```
(DEFUN TOTAL(N)
    (COND (ZEROP N) 0 )
     (T (PLUS N (TOTAL (SUB1 N))))
))
```

論理型言語

- 推論規則「もし～であれば～である」と
  事実「～は～である」の集まり
  でプログラムを記述する
- PROLOG, LOGOなど
- 医学の知識をデータベース化したエキスパートシステムなどの研究

cf. 「知能とは」→ チューリング・テスト
「判定者が隔離された部屋にいる。判定者の隣の部屋には人間もしくはコンピュータがいて、判定者と間接的に「会話」を行うことができる。判定者が相手を「人間」であると感じれば、コンピュータは「知能を持つ」と定義してもよい」

「知能」を定義することの困難さ

ワイゼンバウム：LISPを用いて精神医学者のふりをするプログラム「イライザ」を作成

オブジェクト指向言語

- 複雑なプログラムの開発時に繰り返し現れる「似たようなもの」「似たような処理」を部品化し、再利用が可能になるようにした言語
- 人間の組織や活動や分類に近い自然な概念や抽象的な概念でプログラムを設計し記述できる
- プログラムの生産効率が高まり、大人数による大規模プログラムの製作が容易になった

Javaの例：

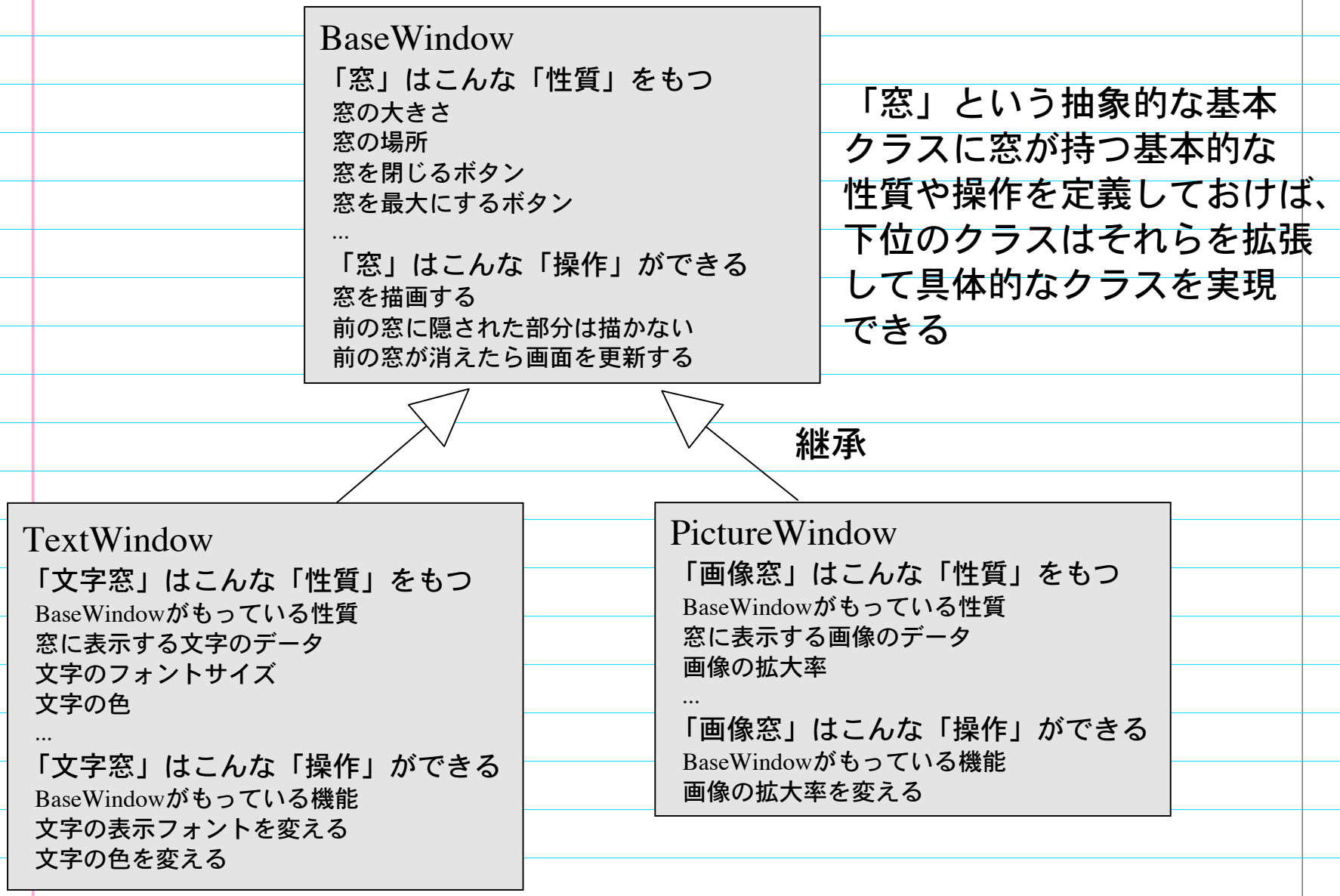
```
import java.applet.*;
import java.awt.*;

public class Kazunowa extends Applet {
  public void paint (Graphics g) {
    int n=15, i, s=0;
    for( i=1; i<=n; i++)
      s+=i;
    g.DrawString( “SUM = “+s, 10, 10);
  }
}
```

出力は「Graphics」というクラスのインスタンス「g」のDrawStringというメソッド(関数)に任せる（どこにどう出力されるかはこのプログラムは知らなくて良い）

他の人が作った「Applet」というクラスの仕様を拡張(継承)して作ることで、Javaプログラムとしての高度な機能をあわせ持つ

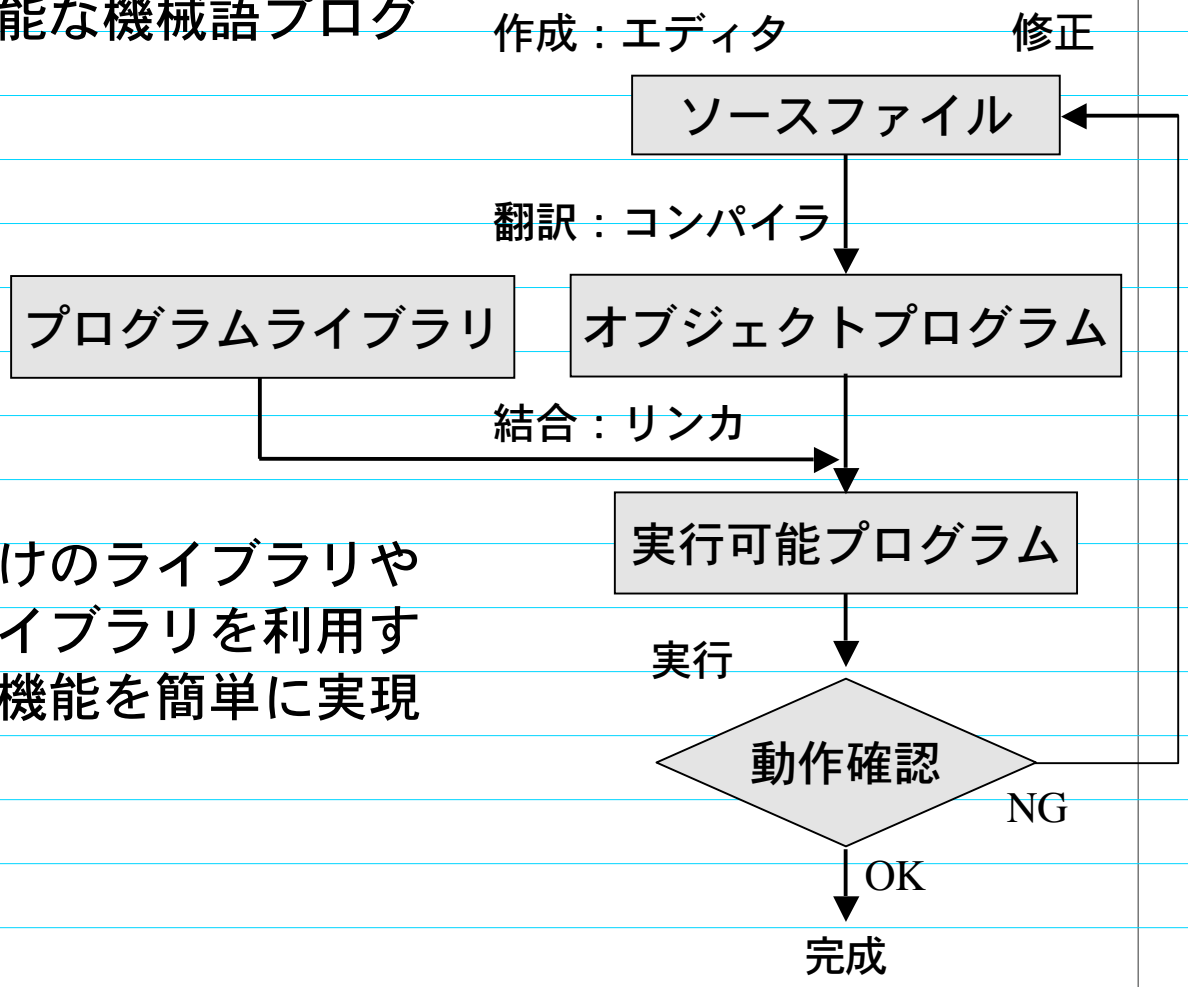
クラス・ダイアグラム



「窓」という抽象的な基本クラスに窓が持つ基本的な性質や操作を定義しておけば、下位のクラスはそれらを拡張して具体的なクラスを実現できる

プログラムの作成から実行まで

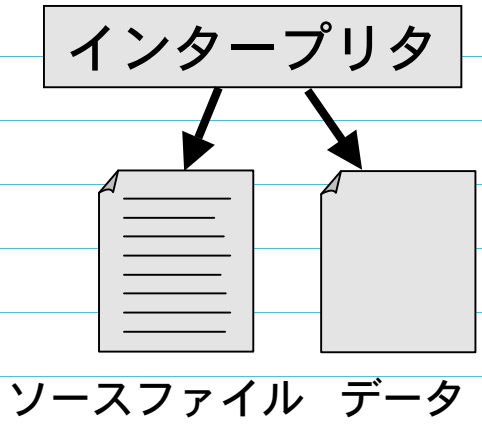
- 人間が読むことのできるソースファイルを「コンパイル」し、「リンク」することで実行可能な機械語プログラムを生成



- それぞれの言語向けのライブラリや他の人が作ったライブラリを利用することで、高度な機能を簡単に実現することができる

コンパイラとインタープリタ

- コンパイラはソースファイルを機械語にあらかじめ翻訳しておく(実行時には機械語プログラムが走る)
- インタープリタはソースファイルから一行ずつ機械語に翻訳しながらプログラムを逐次実行する
- プログラムにループがあると、ループの実行のたびに機械語に翻訳するので速度が遅くなる
- ソースから直接実行できるので手軽かつデバッグが容易



- 自分自身や他のプログラムを改変しながら処理を行うプログラムを作成することが容易にできる
- インタープリタによって処理されるプログラムを「スクリプト」と呼ぶことがある

インタープリタ言語

- Java
  - javacというコンパイラが中間コードを生成し、Javaの実行環境がインタープリタとして中間コードをそのCPUの機械語に逐次翻訳する
  - 移植性、抽象性が高いが実行速度はコンパイラ言語に劣る
- JavaScript
  - HTML言語の中に埋め込んで使う逐次処理言語
- シェルスクリプト
  - UNIXで使うコマンドなどを組み合わせて、複雑なファイル処理やシステム管理作業などをプログラミングできる
  - Bourne-shellスクリプトとcsh(シーシェル)スクリプトがある
- Perl
  - 文字列処理などに優れたライブラリを持つC言語に似たスクリプト言語
  - UNIXユーザーに熱狂的なファンが多く、ソフトウェアの蓄積が豊富
- 表計算ソフト
  - Excelのセルに入れる計算式やVBAマクロなど

プログラム作成の方法論

- 問題の分析、対象の限定、曖昧さの排除、入出力の規定
- アルゴリズムとデータ構造の設計、段階的詳細化、モジュール化
- 処理システムの選定、コーディング
- テスト、結果の検証、不具合の修正
- システムの運用と保守、文書化、マニュアル作成

プログラム作成作業の実際

- エディタの使い方
  UNIX/Linux: emacs, vi など
  Windows, Mac: テキストエディタ、
  あるいはVisual C++などの統合開発環境
- プロジェクト管理
  UNIX/Linux: Makefile
- バージョン管理：ソースファイルの変更の履歴の記録
  CVSなど
- デバッガ
  「バグ」とは
  入力ミス、アルゴリズムのミス、メモリ管理のミスなど
  プログラムを一行ごとに逐次実行しながら変数の値の変化などを調べるツール: gdbなど
- Murphyの法則
  「失敗する可能性のあるものは、必ず失敗する」
- 全ての組み合わせについてテストされていないプログラムにはほぼ全てになんらかのバグがある →「ソフトウェア危機」

プログラミングの例

- 問題の定義：
  Mandelbrot図形を描くプログラムを作ってみたい

- 対象の限定：
  特殊な装置や高価なソフトを買わなくても使えるようにしたい
  一度作ったプログラムをいろんなOSや環境で使えると便利
  それぞれの画面は数秒～数十秒で描画できればよい
  （「毎秒10コマ」のような高速性は要らない）

- 入出力の規定
  カラー図形をWebの画面に描く

- 処理システムの選定：
  Javaアプレットを作ってみる→ Windows, MacOS, Linux,
  i-mode携帯電話, PDAなどで表示が可能になる
  Webブラウザの内部で実行されるので、処理は若干遅い
  マウスと画面だけを使うので標準的なクラスライブラリが使える

Mandelbrot図形とは

- フラクタル図形の一種
  - 図形の一部分を細かく拡大していくと元の図形と似たような図形がどこまでも際限なく現れる
- 複素関数 $f(z) = z^2 + C$ の漸化式

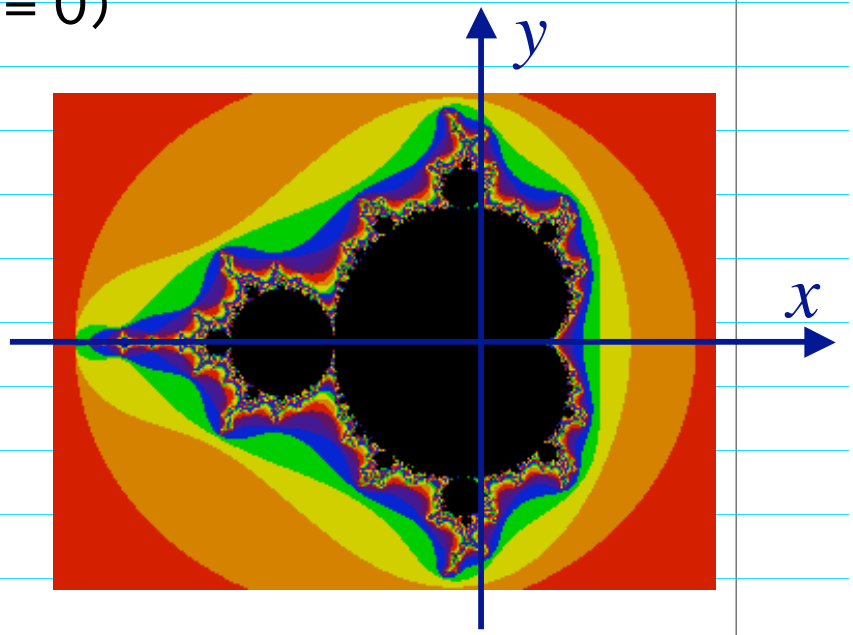
$$z_n = f(z_{n-1})$$

が有限に収束する（$|z_n| \to \infty$ とならない）ような複素数 C の集合（ただし $z_0 = 0$）

- 画面を複素平面に見立てて、それぞれのピクセル座標を複素数 C として扱い

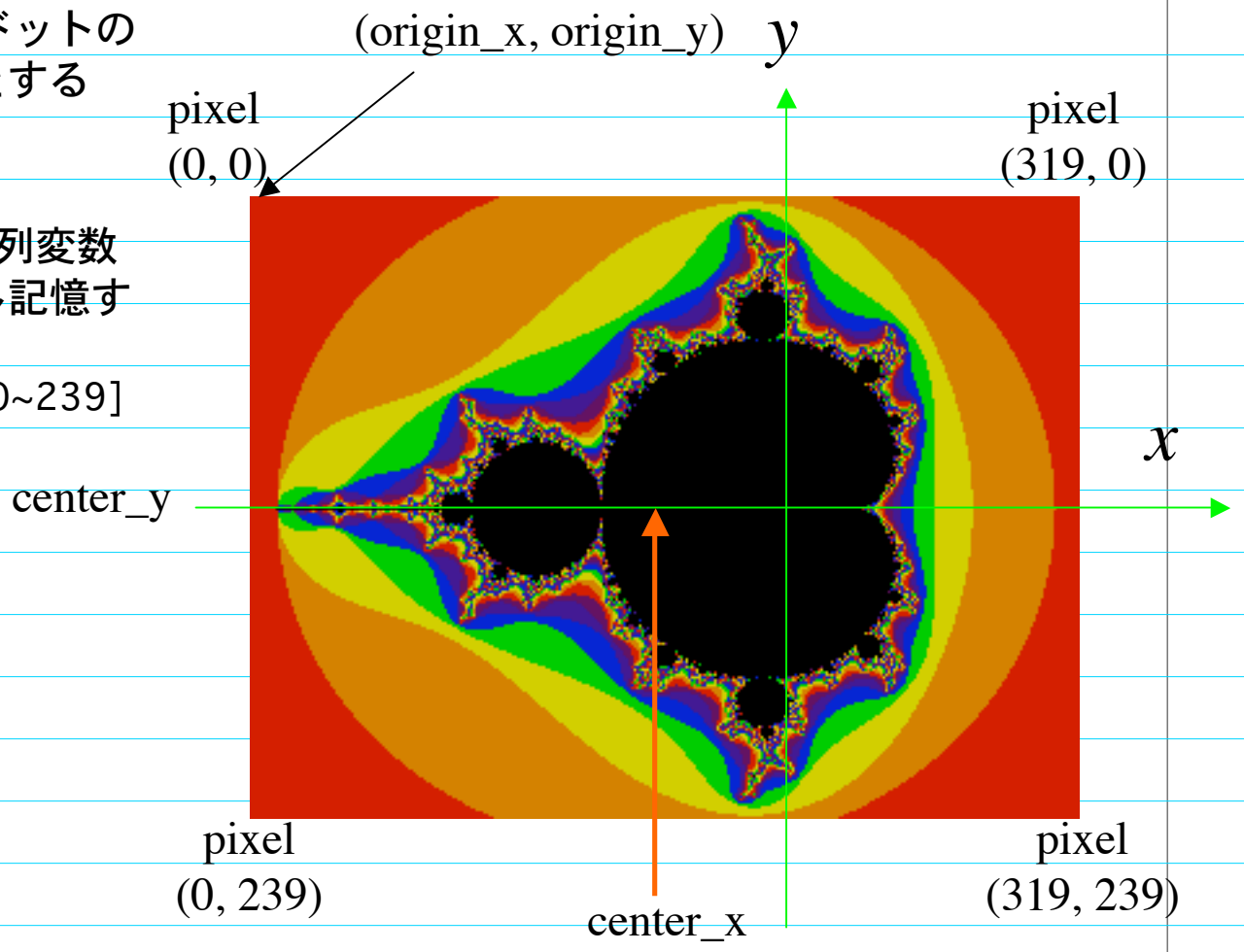
$$z_n = {z_{n-1}}^2 + C$$

が有限回の計算で収束するかどうかを見る。



mandelbrot.javaの製作

- データ構造の設計：
  - 320ドット x 240ドットの画面をキャンバスとする
    pixels_x = 320,
    pixels_y = 240
  - 画素数分の2次元配列変数に収束回数を計算し記憶する
    pixel_data[0~319][0~239]



zoom = 1 → 1 pixel = 0.01
zoom = 10 → 1 pixel = 0.001

漸化式の計算

```
public void create_mandelbrot(){
        int i, j, k;
        float point_x, point_y, value_x, value_y, square_x, square_y;
        for (i=0;i<num_pixels_x;i++) {
            point_x = origin_x + i*step;
            for (j=0;j<num_pixels_y;j++) {
                point_y = origin_y - j*step;
                value_x = point_x;
                value_y = point_y;
                square_x = value_x*value_x;
                square_y = value_y*value_y;
                for (k=0;k<iterate;k++) {
                    value_y = 2*value_x*value_y + point_y;
                    value_x = square_x - square_y + point_x;
                    square_x = value_x*value_x;
                    square_y = value_y*value_y;
                    if (square_x+square_y>4) {
                        break;
                    }
                }
                pixel_data[i][j] = k;
            }
        }
    }
```

画面の描画

```
public void paint(Graphics g){
        int i, j, p_color;

        for (i=0;i<num_pixels_x;i++) {
            for (j=0;j<num_pixels_y;j++) {
                if (pixel_data[i][j]==iterate) {
                    p_color = 0;
                } else {
                    p_color = pixel_data[i][j]%(num_colors) + 1;
                }
                g.setColor(color_value[p_color]);
                g.drawLine(i,j,i,j);
            }
        }
    }
```

クラスの全体構造

```
import java.applet.Applet;
import java.awt.*;

public class mandelbrot extends Applet
{
    int     num_pixels_x = 320, num_pixels_y = 240;
    ....
    int[][] pixel_data;
    Color[] color_value;
    public void init() {
        ....
        create_mandelbrot();
    }
    public void create_mandelbrot(){
        ....
    }
    public void paint(Graphics g){
        ....
    }
}
```

プログラミング上達の秘訣

- 「こんなプログラムを作ってみたい」というモチベーションを持つ
- よい入門書やwebサイトを複数見つける
- プログラム例を目で追いかけるのではなく、必ず全てを打ち込んでみる
- 入力間違いなどがあるとどうなるかを経験する
- 「ここをこうするとどうなる？」という好奇心
- 一つの目的を複数のプログラム言語で書いてみる
- 他の（じょうずな）人が書いたソースファイルを丹念に読み尽くす

「頂上に至る道は一本ではない」

今週のレポート

問1: バブルソートについてインターネットなどで検索してわかったことと、そのページのURLを記せ。

問2:LOGO言語についてインターネットなどで検索して わかったことと、そのページのURLを記せ。

問3: LOGO言語で正三角形を描くプログラムと正六角形を描くプログラムを示せ。

問4:「ソフトウェア危機」という言葉が最初に使われたのはどこで、どのような意味を持つか。インターネットなどで検索してわかったことと、そのページのURLを記せ。

番外：mandelbrot.javaを自分で作ってみたい人はメールの件名を「mandelbrot.java希望」として送って下さい。（レポートのメールとは別に送ること） 自分なりに工夫をこらしたJavaアプレットを作った人には加点します。

参考文献

- 稲垣耕作 著「コンピュータ科学の基礎」コロナ社
  ISBN4-339-02338-8

- 小舘香椎子, 上川井良太郎, 中村克彦 共著
  「教養のコンピュータサイエンス情報科学入門 第2版」 丸善
  ISBN4-621-04871-6

- 菊沢正裕, 山川修, 田中武之共著「情報リテラシー：メディアを手中におさめる基礎能力」森北出版
  ISBN4-621-04871-6