#7 情報・データ・プログラム・バグ

Yutaka Yasuda, 2004 spring term

ソフトウェアの安全性

- 1999 米国での報告
  - 「国民が脆弱なソフトウェアに頼らざるを得ない状況にある」
  - PITAC（大統領情報技術諮問委員会）
- 事例
  - みずほ銀行システムトラブル（バグ）
  - ウィルス問題（セキュリティホール）
  - 470万人顧客名簿流出（情報管理）

ソフトウェアの安全性

- 2002 米国での試算
  - 「ソフトウェアの低品質によって米国経済は 600 億ドル(7.2兆円)、GDP の 0.6% の損失をこうむっている」
  - NIST (National Institute of Standards Technology)
  - 日本の GDP から推定すれば 3 兆円に相当
- バグ
  - ソフトウェア障害の 3/4 が既知の初歩的なプログラミングの誤りによる - Eugene H. Spafford

バグ

- なぜバグは発生するのか？無くならないのか？
  - 「注意深く設計すれば良かった」だけか？
- 情報処理システムが本来持つ構造問題として理解していきましょう

「情報処理」について

情報処理

- 情報処理とはなにか？
- 私たちが目にしているのは実はデータ処理
- コンピュータはデータ自動処理機械である
- なぜデータ処理機械で情報処理が可能なのか？
- 情報とデータの関係を理解する

情報

A B C

- 「情報とは？」
  - インクのシミと A という文字の違い
  - 物理で記述できない違いこそ「情報」
- 情報
  - ものごとの説明（特徴を抽出したもの）
- データ
  - 情報を一定ルールで「記述」したもの

データ

- コンピュータは情報とデータを区別する
  - コンピュータは：情報が持つ意味を解釈しない
  - 人間が：データ処理を情報処理として解釈する
- それを前提に：
  情報処理機械とデータ処理機械は等価である
- コンピュータが情報とデータの役割分担を明確にしたとも言える

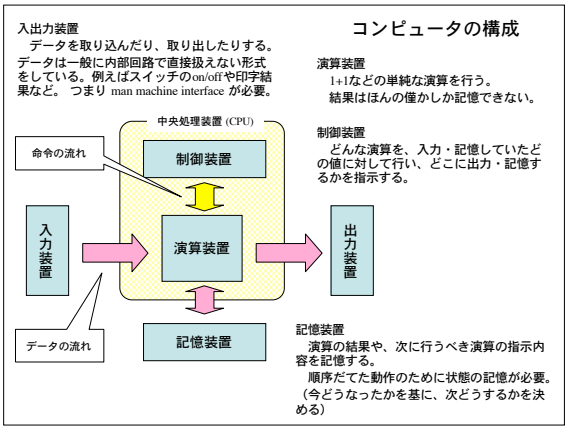
エントロピーという視点からの情報

- エントロピー：無秩序性の尺度
  - より無秩序になるのが自然であり、そちらの方がエントロピーが大きいという概念
- 情報＝エントロピーを減少させるもの
  - 自然の状態（情報が無）
  - これに反する秩序こそが「情報」

情報理論

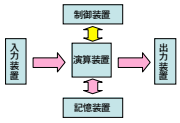
- 1948 シャノン：「情報量」を定義
- エントロピーの視点から
  - データ系列を生成する情報源の確率・統計的な構造から決まるエントロピーをその情報源の情報量という
  - データ量ではく、このデータ系列が担っている真に有効な情報の量を情報量という
  - 生成されたデータを最も効率よく（冗長性や無駄なく）二値系列に直し、最小の長さで記述したときのビット数
- 情報量の概念は符号化によってはじめて明確な意味をもった

具体的なコンピュータの中での データ処理のすがたについて



コンピュータの構成要素

- 現代的コンピュータの構成要素
  - 入出力装置＋演算装置＋制御装置＋記憶装置（＋自動化）
    - データを入力し、
    - （単純な）演算をほどこし、
    - 結果や状態を保存しながら、
    - 繰り返しや条件分岐など順序だてた処理を行い、
    - 結果としてデータを出力する
  - ノイマン型
    - 1946 ノイマンらが提案した構成モデル
    - 図はノイマン型コンピュータの論理的モデル（厳密には Stored Program （後述）であることも含むべき）



(復習) 二進での計算

- 簡単な処理の繰り返しで実現
- 自動計算機械はできた
- 自動データ処理機械はどのようにして？

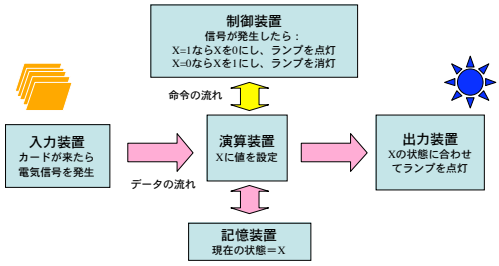
```
234 (11101010) +456 (111001000) =690

        1   1   1   0   1   0   1   0
        +   +   +   +   +   +   +   +
    1   1   1   0   0   1   0   0   0
    =   =   =   =   =   =   =   =   =
    1  10  10   1   0  10   0   1   0
   +1  +1          +1
1   0   1   0   1   1   0   0   1   0
```

- 処理手順はどこに？

データ処理機械を作る （カードの枚数が奇数か偶数かを判定する）

- カード枚数が奇数か偶数か、という「情報」を、カードの通過という現象に変え、
- カードの通過という現象をデータ（電気信号）の発生に変えて、
- データ入力ごとに奇偶を記録する。

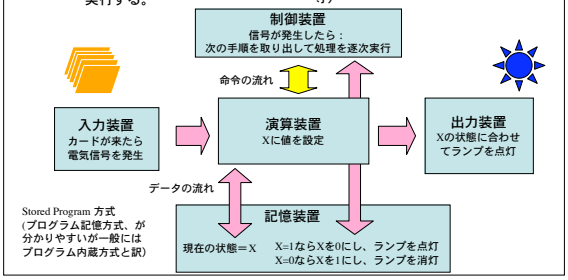
•注目するべき点
- 条件によって動作が左右される
- 過去の状況によって動作が変わるため、記憶装置を利用（過去の状態を更新し続ける）



データ処理機械を作る （カードの枚数が奇数か偶数かを判定する）

- 改善点
  - 処理手順をデータとして表現し、
  - 記憶装置に記録し、
  - 必要に応じて逐次読み込んで実行する。

•つまりプログラムもデータとして扱う
- 処理手順の複雑化に対応しやすい。
- 処理手順を簡単に入れ替え可能（まずカードに処理手順を記録してそれを読ませて記憶装置に記録する等）



Stored Program 方式（プログラム記憶方式、が分かりやすいが一般にはプログラム内蔵方式と訳）

自動情報処理機械

- 自動計算機械はできた
- 自動データ処理機械はできた
- 自動情報処理機械はどのようにして？

- データ処理を情報処理と解釈できるでは？

- 汎用性
  - 計算機がなぜワープロになれるのか？

ソフトウェアとバグの関係

バグ

- プログラムに含まれる「間違い」
- データは意味をもたない
- コンピュータは意味を扱わない

- バグ
  - 無意味な（矛盾した）処理でも指示通り動作する
  - 例えば「金利と残高を加算する」ところを間違えて「年齢と金額を加算」させてしまうかもしれない

- なぜ間違えた指示が最後まで？

プログラミング

- 「1から10までの数を足した結果を出せ」
  - これはコンピュータにとって「難しすぎる」指示
- 手順の明記
  - 「X を 1 から 10 まで変化させ、それを毎回 Y に繰り込め」
- プログラムとは何か
  - 目的に対して「どうやるのか」を詳述したもの

- 意味の消失

「コンピュータはデータの意味を理解しないのと同様に、プログラムの意味も知らない」
（ただ手順だけを知っている）

手順をどのように書くか

- コンピュータは日本語を理解できない
  - 「X を 1 から 10 まで変化させ、それを毎回 Y に繰り込む」のもコンピュータには複雑すぎる
  - コンピュータが理解できる言語で書きくだす必要がある

C言語での例

```
j=0;
for(i=1;i<=10;i++) {
  j=j+i;
};
```

jははじめ 0 だと設定している

1から10まで変化させるということを、「1 からはじめて 10 以下の場合は終わりまでの処理を行い、1 加算してもう一度繰り返し」という表現で明記している

j に増え続ける i を足したものを再び j に代入

手順をどのように書くか

- しかしC言語でもハードウェアには直接理解できない
  - さらに単純な処理に分解しなければ
  - CPU(例は Motorola 68000)向けのアセンブリ言語で記述

```
    st 0,$1004
    mov 1,%o0
    st %o0,$1000
.LL2:
    ld $1000,%o0
    cmp %o0,11
    bgt .LL3
    ld $1004,%o0
    ld $1000,%o1
    add %o0,%o1,%o0
    st %o0,$1004
    ld $1000,%o1
    add %o1,1,%o0
    mov %o0,%o1
    st %o1,$1000
    b .LL2
.LL3
```

元のc言語プログラム

```
j=0;
for(i=1;i<=10;i++) {
  j=j+i;
};
```

ソフトウェアの複雑さ

- アセンブリ言語でもまだ直接は理解できない。
  - 更に機械向けの言語＝機械語に直さなくては

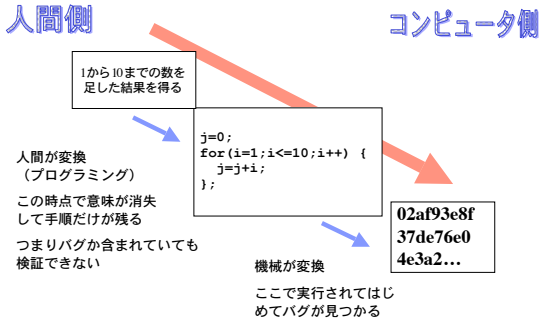
（以下は機械語の一部）

```
7400 5f5f 4354 4f52 5f4c 4953 545f 5f00
5f65 6e76 6972 6f6e 005f 656e 6400 5f47
4c4f 4241 4c5f 4f46 4653 4554 5f54 4142
4c45 5f00 6174 6578 6974 0065 7869 7400
....
```

情報処理から機械語へ

- 処理からプログラムまでの変換過程
  1. 「1から10までの合計」という処理を
  2. どのようにして計算するかという手順に分解
  3. それをプログラミング言語で記述し
  4. 機械語まで変換
- 目的から手順列への変換
  - この作業をプログラミングと呼ぶ
- プログラミング言語から機械語への変換
  - ハードウェアに解釈可能な動作指示へ

二つの変換過程



失われれる意味

- プログラミングの過程で、プログラムから意味は失われる
  - そのプログラムの目的を知るのはプログラマだけ
- コンピュータは情報ではなくデータだけを扱う
  - データが表現する情報は入出力の前後にいる人間が扱う
- ソフトウェアは作業から意味を抜いた手順だけを扱う
  - その意味（内容）は結果を受け取る人間だけが知る
- バグ（意図しないプログラムの振る舞いをひきおこす不具合）が発生する根本的要因のひとつ

失われる意味

- ソフトウェア開発現場での問題
- 分業による意思疎通の壁

対策

- ソフトウェア工学の可能性
  - これは工学の出番ではないのか
- プログラミング段階でのバグ検出
  - 仕様記述
  - プログラミング時に矛盾チェックが可能になる情報をまず記述
- チェックの自動化
  - プログラム脆弱性の検査
  - テストによる実行時矛盾の検出