#5 情報 -データ-プログラム -バグ

Yutaka Yasuda, 2003 spring term

情報処理とは何か

- 情報とデータ
- ハードウェアとソフトウェア
  - コンピュータの基本的な構造
  - プログラム、データ、バグ
- 汎用性、万能機械としての性質
  - なぜ計算する機械がワープロになるのだろう
- これらの要素とその関係を理解しましょう

情報とデータ

- 情報
  - ものごとの説明＝特徴を抽出したもの
  - 実体としての記述（記録）が必要
- データ
  - 情報を記述したもの
- 情報の視点から「ABCDEFG」という記述を見ると
  - 「そこに"A"と書かれている」ことが重要
  - 「黒インクがこのように染みている」ことは重要でない
- 「A」という文字と何かの染みの違い＝情報
  - 物理的には同じ（物理法則では違いを記述できない）
  - 情報という視点からは違う（その違いこそ「情報」）

データ

- コンピュータは情報とデータを完全に区別する
- 情報が持つ意味をコンピュータは解釈しない
  - 情報から意味を完全に除去し、表現だけを残してデータとし、それに基づいて処理を行っている
  - データ処理を、情報処理としてとらえる
- 情報処理機械とデータ処理機械は等価である
  - 人間は情報をデータとしてコンピュータに入力する
  - 出力されるものもまたデータ（もしくはデータの変形）であるが、人間はそこから情報、そして意味を取り出す
- 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

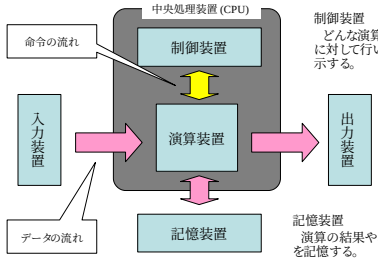
- エントロピー：無秩序性の尺度
  - より無秩序になるのが自然であり、そちらの方がエントロピーが大きいという概念
    - 物理学（物質＝エネルギーで世界を構成するという概念）の世界で誕生
    - エネルギー(例えば熱)は高い状態から拡散し、全体に低い状態へと移行する、という自然の性質を記述するために用いられる
    - その後、様々な分野に導入されている
- 情報＝エントロピーを減少させるもの
  - 自然の状態（情報が無）に反する秩序＝情報

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（その情報源はどの程度の秩序を生み出したのか）
  - 単位はビット/秒 または ビット/文字と定義する
- データ量ではなく、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

コンピュータの構成

入出力装置
　データを取り込んだり，取り出したりする。
データは一般に内部回路で直接扱えない形式をしている。例えばスイッチのon/offや印字結果など。
つまりman machine interface が必要。



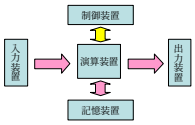
演算装置
　1+1などの単純な演算を行う。
　結果はほんの僅かしか記憶できない。

制御装置
　どんな演算を、入力 記憶していたどの値に対して行い、どこに出力 記憶するかを指示する。

記憶装置
　演算の結果や、次に行うべき演算の指示内容を記憶する。
　順序だてた動作のために状態の記憶が必要。
　(今どうなったかを基に、次どうするかを決める)

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

- 10進で3桁の足し算を分解
  - 10進1桁の足し算を三回 (繰り上がり込み)
- 2進では9桁、足し算も9回

234+456=690 は?

```
  2   3   4
  +   +   +
  4   5   6
  =   =   =
  6   8  10
         +1
  6   9   0
```

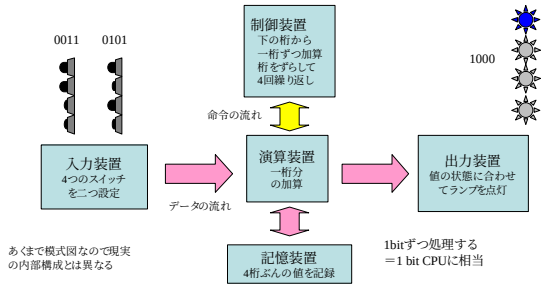
234 (11101010) +456 (111001000) =690

```
      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
```

計算機械を作る (1bit CPU で二進数4桁の足し算を行う例)

- 3, 5 という数値二つの 情報」を
- 二進四桁、4 bit x 2 のデータ (電気信号)に変えて、
- 一桁ずつ順ぐりに四桁の加算を行い
- 結果をランプを点灯させることで表現

- 注目するべき点
  - 単純な演算装置を繰り返して使う
  - 四桁分の結果を保存するために記憶装置を使う



あくまで模式図なので現実の内部構成とは異なる

1bitずつ処理する
＝1 bit CPUに相当

複雑な計算の実現

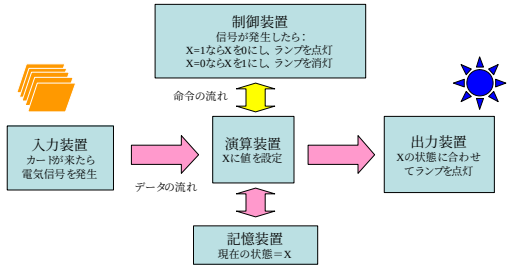
- 高い精度の演算
  - 少ない桁数の演算を繰り返すことで処理
- 複雑な計算
  - 演算可能な単位に分解し、
  - 自動的に少しずつ手順を踏んで処理
- 手順に従った演算の連続で全てを処理
  - コンピュータによる計算処理の実体
- 自動計算機械、自動処理機械の誕生

(一般的) コンピュータの処理対象の限界をも示す
–単純な処理に分解できない仕事には対応できない
–多くの場合、分解できない＝よく分かっていない

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。

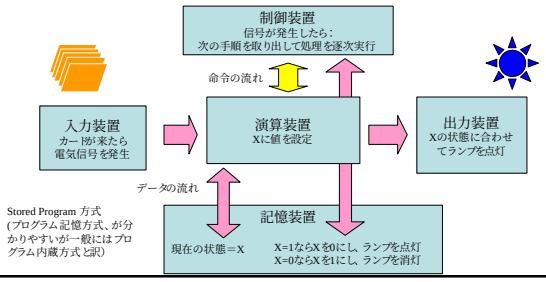
- 注目するべき点
  - 条件によって動作が左右される
  - 過去の状況によって動作が変わるため、記憶装置を利用 (過去の状態を更新し続ける)



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
- 自動情報処理機械と解釈できるではないか
  - データ化
    - 情報を符号化し、機械はデータだけを扱えばよい
  - 万能（汎用性）
    - どのような情報でも符号化できれば処理できる
    - 非常にコンピュータが普及している理由
    - 非常に多様な目的に適用されている理由
- 「データ処理機械がワープロになれる」理由

ハードウェア、ソフトウェア、プログラム、データ

- コンピュータ
  - ハードウェア+ソフトウェア
- ハードウェア
  - 計算処理を実施し、状態を記憶する実体
- ソフトウェア
  - (おおよそ)プログラムとデータからなる
    - 計算機を動かしているハードウェア以外のものすべてと考えても良い
  - プログラム：(典型的には)動作手順
  - データ:情報の表現
    - プログラムによる処理の対象として入出力されるもの

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
  - 人間だけが知っている
- コンピュータは意味を扱わない
  - コンピュータに「間違い」「正しい」という概念はない
- 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
  - 結果から処理間違いを発見して修正するのは非常に困難

ソフトウェアの複雑さ

- 「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
  - 「何をすればよいか」を考えることはできない
  - 「何をどうやるのか」まで微細に明記して与える
- プログラムとは何か
  - 目的に対して「どうやるのか」を記述したものがプログラムであり、その記述作業がプログラミングである。
  - 日本語的プログラム
    Xを1から10まで変化させ、それを毎回Yに繰り込む
  - 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)

ソフトウェアの複雑さ

- 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、「i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

ソフトウェアの複雑さ

- しかしC言語でもコンピュータのハードウェア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 を仮定)向けのアセンブラで記述

```
      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
```

元の記述

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

ソフトウェアの複雑さ

- アセンブラ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
    - アセンブラ一行が数バイトの機械語に変換される
  - 例は機械語の一部）

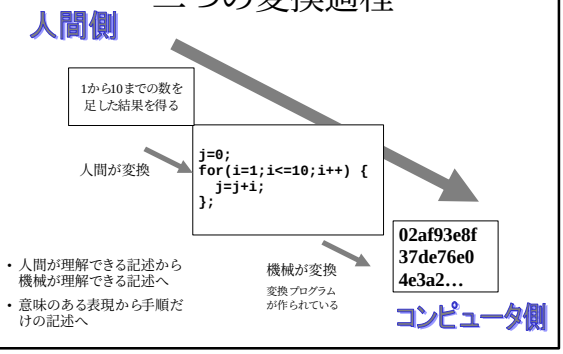
```
7400 5f5f 4354 4f52 5f4c 4953 545f 5f00
5f65 6e76 6972 6f6e 005f 656e 6400 5f47
4c4f 4241 4c5f 4f46 4653 4554 5f54 4142
4c45 5f00 6174 6578 6974 0065 7869 7400
....
```

- 日本語で一行だった処理が大量の手順列に変換された

二つの変換過程

- ソフトウェアに存在する二種類の変換
- 目的から手順列への変換
  - 人間が行う
  - この作業をプログラミングと呼ぶ
  - コンピュータが (間接的にでも) 理解できる言語で記述
- プログラミング言語から機械語への変換
  - ハードウェアの機能を直接呼び出す指示列へ
  - 大抵の場合）機械が行う

二つの変換過程



- 人間が理解できる記述から機械が理解できる記述へ
- 意味のある表現から手順だけの記述へ

失われれる意味

- 目的から手順列への変換過程で、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だけが知る
- 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
- より複雑な処理のために
  - ブレイクスルーが求められている