第６章　問題の解き方

レポートをコピペで済ませた皆さんも頑張って理解してください

【アルゴリズム】

◆アルゴリズムの役割

コンピュータによって問題を解く

… 人間が書いたプログラムをコンピュータが実行するということ

アルゴリズム … プログラムになる前の段階の計算手順

◇現実問題からプログラムを作る過程（図 6-1）

①問題をモデル化する

②モデル化された問題に対して、それを解く計算手順（＝ アルゴリズム）を考える

③手順通りに計算するプログラムを作る

図 6-1

◇アルゴリズム≠プログラム

前章で登場した八十八夜問題の解法はまだプログラムではなく、アルゴリズム段階

```
<残り日数> ← 4＋87
m ← 2
while <残り日数> > daymonth_m do
  <残り日数> ← <残り日数> − daymonth_m
  m ← m＋1
done
m"月"<残り日数>"日"と表示
```

図 6-2

これを java などのプログラミング言語で書き直すと、

あとは計算を実行するだけでよいプログラムになる

アルゴリズムの選択がプログラムの実行時間を大きく左右する

アルゴリズムとプログラムを

ひっくるめて考える（頭の中で解法を考えながらプログラムを作る）と…

→ プログラムを作ってから新しい解法を思いついたら二度手間

→ いちいちまたプログラムを打ち直し

→ アルゴリズムの時点で比較、選択した方が効率がよい

◆アルゴリズムの実例

◇コンピュータより古いアルゴリズム

ユークリッドの互除法、ドント方式など

◇x の平方根を求める問題

> ある正の実数 x が与えられたときに、２乗すると x に近くなる正の実数 y を精度 δ で求める。つまり、$|\sqrt{x} - y| < \delta$ となるような y を１つ求める。

▽**反復法**を用いたアルゴリズム

y ← 0
while $(y+\delta)^2 < x$ do
  y ← y+ δ
done
return y

※「return y」は「y を解として計算を終了」の意

▽反復法を用いたアルゴリズムの考え方

→ 解の候補を順に検討 → 候補を２乗し、それが x を超えたら１つ前の候補が解

δ＝0.5 で x＝8 の場合

→ y＝0 としよう

… 0 の２乗は 0 で、0 ＜ 8 → y に 0 を代入しといて次へ

→ y＝0.5 としよう

… 0.5 の２乗は 0.25 で、0.25 ＜ 8 → y に 0.5 を代入しといて次へ

→ … → y=2.5 としよう

… 2.5 の２乗は 6.25 で、6.25 ＜ 8 → y に 2.5 を代入しといて次へ

→ y=3 としよう

… 3 の２乗は 9 で、9 ＞ 8 → y に 3 は代入せず、反復処理を終える

→ return y すなわち、2.5 を解として計算を終了

▽反復法を用いたアルゴリズムの性質

2 の平方根を精度 δ＝0.0001 で求めると、14142 回の計算が必要（図 6-3）

回数	0	1	2	…	14140	14141	14142
候補(y)	0.0000	0.0001	0.0002	…	1.4140	1.4141	1.4142
$(y+\delta)^2$	0.00000	0.00000	0.00000	…	1.99968	1.99996	2.00024

図 6-3

精度に逆比例した必要計算回数

→ １桁よい精度の解を求めるには 10 倍の計算回数がかかる

▽**二分法**を用いたアルゴリズム（x ＞ 1 を条件とする）

a ← 0

b ← x

while b－a ＞ δ do

$$c \leftarrow \frac{a+b}{2}$$

if c^2 ＞ x then b ← c else a ← c endif

done

return a

▽二分法を用いたアルゴリズムの考え方

数直線上で、解の含まれる区間をどんどん絞っていく

区間を半分に分け、左と右、解の含まれる方を次の区間にする

上のアルゴリズムではこの区間の左端（＝最小値）が a、

右端（＝最大値）が b という変数で表されている

つまり、a が大きくなり、b が小さくなる程区間は絞られていく

この区間の大きさが精度 δ よりも小さくなったところで計算を終え、

a（＝区間の最小値）を解として終了

図 6-4 は x=2 のときの例

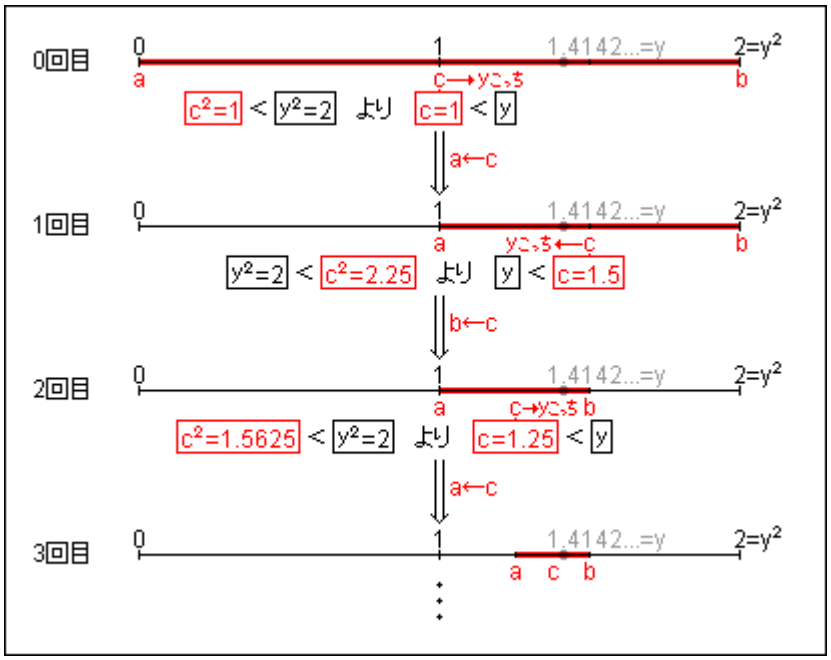

図 6-4

▽二分法を用いたアルゴリズムの性質

2 の平方根を精度 δ ＝0.0001 で求めると、15 回の計算が必要

→ 反復法よりも効率的

【計算量】

◆計算量の考え方

計算量 … アルゴリズムをもとにしたプログラムの実行時間を見積もるための指標

明日の天気を知るプログラムがあっても、実行するのに１週間かかっては意味がない

→ 事前に計算量を見積もる

→ 無駄なプログラムを作らなくて済む

より効率的なプログラムが選択できる

◇**計算量のオーダー** … 計算量の見積もりにおいて基準とされる、非常に大まかな尺度

どんな変数に比例するかのみが重要で、**定数は無視**される

▽N 個のデータを処理するプログラムＡとＢ

Ａ … 2 秒の処理を N 回繰り返し、最後に 3 秒の処理を行う

→ プログラムの実行にかかる時間は 2N＋3 秒

→ 計算量のオーダーは N

Ｂ … 10 秒の処理を $\log_2 N$ 回繰り返す

→ プログラムの実行にかかる時間は $10\log_2 N$ 秒

→ 計算量のオーダーは $\log_2 N$

▽定数を無視する理由

処理１回当たりの計算時間の違いよりも、

計算回数が何に比例するかの違いの方が、全体の実行時間に決定的な違いを及ぼす

→ N＝10 のとき　Ａ → 23 秒　Ｂ → 約 33 秒

N＝1000 のとき　Ａ → 2003 秒　Ｂ → 約 100 秒

N＝10 万のとき　Ａ → 20 万 3 秒（約 56 時間）　Ｂ → 約 166 秒（約 3 分）

… この大きな差は 2 と 10 の定数から生じたものではなく、

N と log2N という違いから生じたもの

→ N が大きくなればなるほど違いが大きくなる

また、定数倍はコンピュータの性能などによっても左右されてしまう

→ 10 秒かかるはずが 9.7 秒しかかかってなかったり

◆計算量の例

◇x の平方根を求めるアルゴリズムにおける計算量（のオーダー）

反復法 → $\sqrt{x}\ /\ \delta$

二分法 → $\log_2(x/\delta)$（＝ $x/2^n < \delta$ となるような最小の n）

◇フィボナッチ数を求めるアルゴリズムにおける計算量（のオーダー）

アルゴリズム自体はテスト範囲じゃないので、「こんなのもあるんだ」程度に

再帰アルゴリズム → 2^n

メモ化アルゴリズム → n

◇データの個数 n に対する計算時間の、アルゴリズムによる違い（図 6-5）

1 回の処理に 1 ns（ナノ秒）かかる場合

n	1	10	100	1000	1万	10万	100万	1000万
log n	—	1ns	2ns	3ns	4ns	5ns	6ns	7ns
n	1ns	10ns	100ns	1μs	10μs	100μs	1ms	10ms
n^2	1ns	100ns	10μs	1ms	0.1s	10s	16分	27時間
n^3	1ns	1μs	1ms	1s	16分	11日	31年	3000年
2^n	2ns	1μs	40兆年					

図 6-5

第７章　コンピュータの仕組み

前半は訳のわからん用語の暗記物、後半はややこしい演算、頑張りましょう

【計算の実現機構】

◆プログラム内蔵方式

プログラム … コンピュータの実行する計算処理手順に関わる情報

データ … コンピュータが処理する対象の情報

万能チューリング機械 … プログラムとデータをテープに記録する仮想的な計算機械

プログラム内蔵方式 … テープに代わってメモリ上にプログラムとデータを保持し、
プログラムに従って計算を進めるような機構
現実のコンピュータはこの方式をとる

フォンノイマン型コンピュータ … プログラム内蔵方式のコンピュータの別名

◆コンピュータの基本構成（図 7-1）

演算装置 … データに対して計算処理を施す演算装置

主記憶装置（メインメモリ） … 複数のデータやプログラムを記憶する装置

制御装置 … 主記憶装置上のプログラムに従って演算装置を駆動する装置
主記憶装置へのデータの読み書きも担当

中央処理装置（CPU） … 制御装置＋演算装置　Central Processing Unit

マイクロプロセッサ（MPU） … CPU を１つの半導体集積回路として実現したもの
Micro Processing Unit

アドレス … 主記憶装置上の情報の位置を表す数値

演算レジスタ … 中央処理装置による計算に使用するデータを一時的に保持する装置
初期のコンピュータではこれが１つだけしかなく、
アキュムレータ（AC）と呼ばれていた

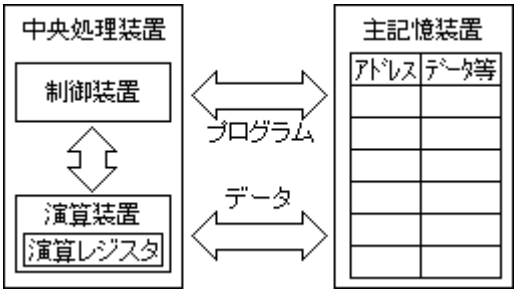

図 7-1

※演算レジスタが十分多く存在すると、計算の際に主記憶装置にほとんどアクセスせずにすむ。それでは勉強にならないので、この授業では演算レジスタは１つという設定にする。

◆機械語レベルのプログラム例

命令集合 … 個々の CPU で利用できる命令の集まり（図 7-2）

命令コード … 命令の種類を表す符号

オペランド … 命令コードの後につけられる付加情報

種類	内容	意味
データ転送命令	load A	アドレスAのデータを演算レジスタに読み込む
	store A	演算レジスタのデータをアドレスAに書き込む
演算命令	add A	アドレスAのデータを演算レジスタの値に加える
	subtract A	アドレスAのデータを演算レジスタの値から引く
分岐命令	jump A	アドレスAにプログラムの実行を移す
	jumpzero A	演算レジスタの値が0のときのみjump Aと同じ機能
その他	write	演算レジスタのデータを出力する
	halt	プログラムの実行を停止する

図 7-2

データ転送命令 … 主記憶装置上のデータを演算レジスタに読み込み
演算レジスタのデータを主記憶装置に書き込み

演算命令 … 算術演算（四則演算）や論理演算（AND や OR など）を実行
結果は演算レジスタに書き込まれる

分岐命令 … 指定したアドレスにプログラムの実行を移す
条件分岐（ジャンプ）と無条件分岐（ジャンプ）がある

条件ジャンプ … 演算レジスタの値に応じて選択的に実行を移す（jumpzero）

無条件ジャンプ … 演算レジスタの値にかかわらず実行を移す（jump）

上の命令集合では、すべての命令はオペランドとしてたかだか 1 つの値をとる
… add A B などの形式はとれない

◇load と store（図 7-3）

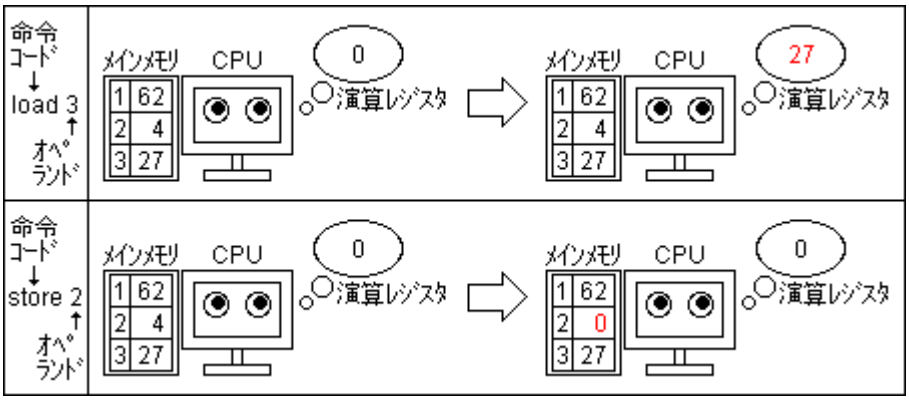

図 7-3

◇add、jumpzero、write（図 7-4）

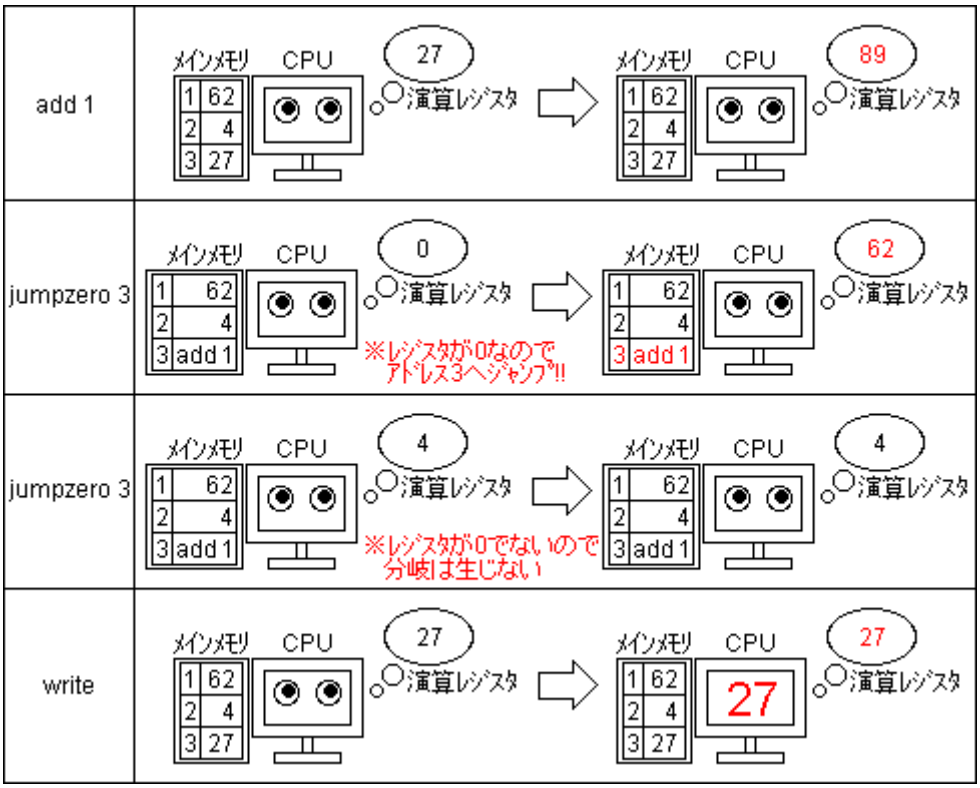

図 7-4

◇1 から 10 までの和を求めるプログラム

アドレス	命令	アドレス	命令
1001	load 2001	1008	jump 1001
1002	add 2002	1009	load 2001
1003	store 2001	1010	write
1004	load 2002	1011	halt
1005	subtract 2003	2001	0
1006	store 2002	2002	10
1007	jumpzero 1009	2003	1

▽各アドレスの意味

1001～1011 には命令（プログラム）が記憶されている

2001～2001 にはデータが記憶されている

→ プログラムとデータがぐちゃぐちゃになってややこしくならないように

アドレス 2001 … 和を記憶する場所

0→10→19→27→34→…→55 と増えていく

アドレス 2002 … ここの値が 0 になるまで処理が続く

10→9→8→…→0 と減っていく

アドレス 2003 … アドレス 2002 の値を 1 ずつ減らすための「1」を記憶
ここを 2 にすると 2002 の値は 10→8→6→… と減っていく

▽処理の流れ（**さっきのプログラムと見比べながら進むと理解しやすいかも**）

アドレスの小さい場所の命令から大きい場所の命令へ、順々に処理

以下実際の処理の流れ（[　] 内は順に、演算レジスタ,アドレス 2001,2002 の値）

○1001　アドレス 2001 の値 0 を演算レジスタに読み込む［0,0,10］
○1002　アドレス 2002 の値 10 を演算レジスタの値 0 に加算［10,0,10］
○1003　演算レジスタの値 10 をアドレス 2001 の値 0 に上書き［10,10,10］
○1004　アドレス 2002 の値 10 を演算レジスタの値 10 に上書き［10,10,10］
○1005　演算レジスタの値 10 からアドレス 2003 の値 1 を減算［9,10,10］
○1006　演算レジスタの値 9 をアドレス 2002 の値 10 に上書き［9,10,9］
○1007　演算レジスタの値（9）が 0 ではないので無視［9,10,9］
○1008　アドレス 1001 へ戻る［9,10,9］
○1001　アドレス 2001 の値 10 を演算レジスタの値 9 に上書き［10,10,9］
○1002　アドレス 2002 の値 9 を演算レジスタの値 10 に加算［19,10,9］
○1003　演算レジスタの値 19 をアドレス 2001 の値 10 に上書き［19,19,9］
○1004　アドレス 2002 の値 9 を演算レジスタの値 19 に上書き［9,19,9］
○1005　演算レジスタの値 9 からアドレス 2003 の値 1 を減算［8,19,9］
○1006　演算レジスタの値 8 をアドレス 2002 の値 9 に上書き［8,19,8］
○1007　演算レジスタの値（8）が 0 ではないので無視［8,19,8］
○1008　アドレス 1001 へ戻る［8,19,8］
：
○1008　アドレス 1001 へ戻る［1,54,1］
○1001　アドレス 2001 の値 54 を演算レジスタの値 1 に上書き［54,54,1］
○1002　アドレス 2002 の値 1 を演算レジスタの値 54 に加算［55,54,1］
○1003　演算レジスタの値 55 をアドレス 2001 の値 54 に上書き［55,55,1］
○1004　アドレス 2002 の値 1 を演算レジスタの値 55 に上書き［1,55,1］
○1005　演算レジスタの値 1 からアドレス 2003 の値 1 を減算［0,55,1］
○1006　演算レジスタの値 0 をアドレス 2002 の値 1 に上書き［0,55,0］
○1007　演算レジスタの値（0）が 0 なので、アドレス 1009 へ進む［0,55,0］
○1009　アドレス 2001 の値 55 を演算レジスタの値 0 に上書き[55,55,0］
○1010　演算レジスタの値 55 を出力、画面に表示
○1011　プログラム停止

▽高水準言語で記述すると

```
sum ← 0
i ← 10
while i ≠ 0 do
  sum ← sum＋i
  i ← i−1
done
write(sum)
```

※こっちの方が理解しやすい（**はず**）

◆実際の機械語プログラム

ビットパターン（0 と 1 の列）による記述（… バイナリプログラム）

→ コンピュータ内の物理的表現、電圧の高低、電流の on/off に対応

さっきやった和のプログラムは load など英文字が含まれているので、
機械語のプログラムというよりは
アセンブリ言語（シケプリ前編 p.24 参照）によるプログラムに相当

バイナリプログラムでは図 7-2 の８つの命令コードを区別するのに
３（＝ $\log_2 8$）ビットのビットパターンが用いられる（000,001,010,…111）

アドレス、演算レジスタ上のデータ、主記憶装置上のデータ

→ すべて２進数による表現

【論理演算と組み合わせ回路】

ここが一番ややこしいんじゃないでしょうか。もう湯浅とか出てこさせてふざけてる場合じゃないです。ファイト！！

◆**組み合わせ回路**とは何ぞや？

２進符号表現の演算を実現するための記憶をもたない回路
n ビットの入力があると、即座かつ一意に m ビットの出力を返す
要は、何か入力したらすぐ（ある法則に従って）何か返してくれるってこと！
この程度の理解でいいと思います。

◆n 入力 1 出力の論理関数

論理関数 … ２進法に基づく演算

n ビットの入力パターンは 2^n 通り

→ この入力パターンについて、それぞれ 0 か 1 の出力が可能

→ 0 か 1 の２通りが 2^n パターン

→ 論理関数の種類は $\mathbf{2^{2^n}}$ 種類

完備な組み合わせ … 2^{2^n} 通りのすべての論理関数を実現できる演算の組み合わせ

→｛AND，OR，NOT｝の組み合わせが有名

→ **証明は教科書 p.159 にあるけど、わけわからんし覚えなくていいと思います**

※論理演算では 0 のことを偽(FALSE)、1 のことを真(TRUE)と呼んだりします

AND(x,y) … ２つの入力 x,y がともに真(1)のときに真(1)と出力

→ AND(0,0)⇒0　AND(0,1)⇒0　AND(1,0)⇒0　AND(1,1)⇒1

OR(x,y) … ２つの入力 x,y の少なくとも一方が真(1)のときに真(1)と出力

→ OR(0,0)⇒0　OR(0,1)⇒1　OR(1,0)⇒1　OR(1,1)⇒1

NOT(x) … 入力 x が真(1)のときに偽(0)、偽(0)のときに真(1)と出力
つまり真偽が逆転する

→ NOT(0)⇒1　NOT(1)⇒0

◇たとえば 1 入力の場合

論理関数の種類は 4 種類

▽具体的には

①入力 i が 0 でも 1 でも、出力 o が 0 となる論理関数

②入力 i が 0 なら 0、入力が 1 なら 1 を出力(o)する論理関数

③入力 i が 0 なら 1、入力が 1 なら 0 を出力(o)する論理関数

④入力 i が 0 でも 1 でも、出力 o が 1 となる論理関数

▽｛AND，OR，NOT｝を使って表現すると

①o＝AND(i,NOT(i))

②o＝i

③o＝NOT(i)

④o＝OR(i,NOT(i))

②と③はわかりやすいと思います。
入力されたものをそのまま出力するだけだから o＝i
入力されたものの真偽をひっくりかえして出力するだけだから o＝NOT(i)です。
①はどんな入力に対しても偽(0)と出力したい。そこで、AND と NOT を利用します。
x＝i，y＝NOT(i)としましょう。
y は i の真偽を入れ替えたものですから、x の真偽とも入れ替わってます。
つまり x が真のとき y は偽だし、x が偽のとき y は真です。絶対に。
じゃぁ AND(x,y)は？AND(x,y)では x,y がともに真のときだけ真と出力されます。
ところが上で言ったようにこの場合の x と y はともに真ではありえません。
じゃぁもう偽(0)と出力するしかないじゃないですか！
今度は④OR(x,y)を検証してみましょう。
OR(x,y)は x,y 少なくともどちらか一方が真のとき真と出力されます。
逆に言えば x,y がともに偽のときだけ偽と出力されます。
でも、x,y がともに偽ではありえませんよね。
ほら、もう真(1)と出力するしかないじゃないですか！
これが①、④の考え方です。

◇今度は２入力

論理関数の種類は 16 種類（図 7-5）

▽具体的には（x,y を入力する）

①入力が x=y=0 でも、x=0 かつ y=1 でも、x=1 かつ y=0 でも、x=y=1 でも 0 を出力
②入力が x=y=0、x=0 かつ y=1、x=1 かつ y=0 なら 0、x=y=1 なら 1 を出力
：
⑮入力が x=y=0、x=0 かつ y=1、x=1 かつ y=0 なら 1、x=y=1 なら 0 を出力
⑯入力が x=y=0 でも、x=0 かつ y=1 でも、x=1 かつ y=0 でも、x=y=1 でも 1 を出力

(x, y)	(0, 0)	(0, 1)	(1, 0)	(1, 1)
①	0	0	0	0
②	0	0	0	1
③	0	0	1	0
④	0	0	1	1
⑤	0	1	0	0
⑥	0	1	0	1
⑦	0	1	1	0
⑧	0	1	1	1
⑨	1	0	0	0
⑩	1	0	0	1
⑪	1	0	1	0
⑫	1	0	1	1
⑬	1	1	0	0
⑭	1	1	0	1
⑮	1	1	1	0
⑯	1	1	1	1

図 7-5

図 7-5 のように、２進演算の入出力の関係を表にしたものを**真理値表**といいます

▽論理関数表現

①は x,y がどんな入力になっても 0 を出力するという論理関数

→ **0** と表される（これをさっきは AND(i,NOT(i))の形に変形しました）

②は x,y がともに 1 のとき 1 を出力するという論理関数

→ **AND(x,y)**と表される

④は x の真偽をそのまま出力するという論理関数

→ **x** と表される（上の例で言うと o＝i）

⑥は y の真偽をそのまま出力するという論理関数

→ **y** と表される（上の例で言うと o＝i）

⑧は x,y 少なくとも一方が 1 のとき 1 を出力するという論理関数

→ **OR(x,y)**と表される

⑪は y の真偽を逆転して出力するという論理関数

→ **NOT(y)**と表される

⑬は x の真偽を逆転して出力するという論理関数

→ **NOT(x)**と表される

⑯は x,y がどんな入力になっても 1 を出力するという論理関数

→ **1** と表される（これをさっきは OR(i,NOT(i))の形に変形しました）

これらはさっきやりましたね

ここではさらに論理関数表現が増えます

⑦は x,y どちらか一方だけが 1 のとき 1 を出力するという論理関数

→**XOR(x,y)**と表される（eXclusive OR の略 … 排他的論理和）

⑨は x,y がともに 0 のとき 1 を出力するという論理関数

→**NOR(x,y)**と表される

（つまり OR(x,y)の否定です、⑧と比べるときれいに真偽が逆転してますよね）

⑩は x と y が一致するとき 1 を出力するという論理関数

→**EQ(x,y)**と表される（**EQuivalence の略ですが EQual の略って覚えていいです**）

… XOR(x,y)と否定の関係にあります

⑮は x,y 少なくとも一方が 0 のとき 1 を出力するという論理関数

→**NAND(x,y)**と表される（見ての通り AND(x,y)と否定関係にあります）

▽またまた完備性の話

さっきも言ったように｛AND，OR，NOT｝は完備な組み合わせです。

なので、⑦⑨⑩⑮の論理関数表現も、この３つの表現だけに置き換え可能です。

これは後でやります。ちなみに NAND、NOR はそれぞれ単独で完備性を備えてます。

つまり、すべての表現を NAND だけ、または NOR だけで表現可能なんです。

◆MIL 記法

回路に似せた論理関数表現法

基本的な論理関数を**基本素子**（図 7-6）で表し、その間の結線で論理関数を表現

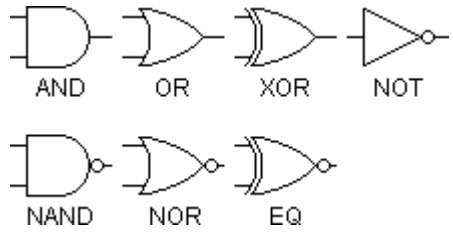

図 7-6

素子の図中の○は、**ビット反転**（0→1、1→0）を表している

◇XOR 回路の表現

では、さっき「後でやる」と言っておいた、

XOR(x,y)の｛AND，OR，NOT｝での置き換えをやってみましょう。

結論から言うと、XOR(x,y)という論理演算は、

OR(AND(x,NOT(y)),AND(NOT(x),y))

という訳のわからん形に置き換えられます。

これを MIL 記法で表してみよう！

まず一番外枠の OR の基本素子を置きます（図 7-7）。

図 7-7

次に、OR への入力は AND(x,NOT(y))と AND(NOT(x),y)、

これは AND 基本素子から出力される値が OR に入力されるということなので、

OR 基本素子の２つの入力端子に AND 基本素子２つの出力端子を繋ぎます（図 7-8）。

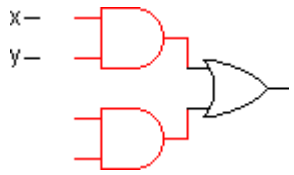

図 7-8

上の AND 基本素子には x と NOT(y)を入力します。

NOT(y)は y をビット反転させたものなので、

入力の直前に○を置いて反転させます（図 7-9）。

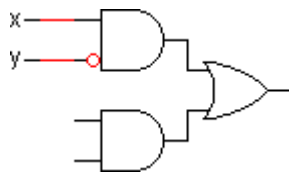

図 7-9

今度は下の AND 基本素子にも入力します。

NOT(x)と y を入力するので、y をそのまま引っ張ってきて、
x の方は○をつけてつなぎます（図 7-10）。

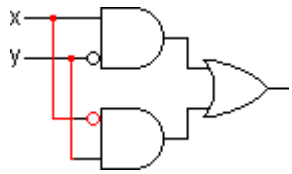

図 7-10

これで完成です（図 7-11）。

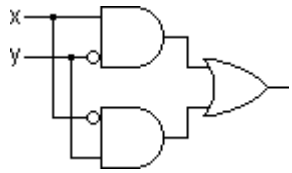

図 7-11

さて、本当に XOR(x,y)としての回路になっているのか確かめてみましょう。
次の表を使って説明します（図 7-12）。

x	y	$\bar{x}$	$\bar{y}$	上の AND	下の AND	OR
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

図 7-12

まず入力パターンとしては(x,y)＝(0,0),(0,1),(1,0),(1,1)の 4 通りあります。
そのそれぞれに対し、(NOT(x),NOT(y))＝(1,1),(1,0),(0,1),(0,0)となります。
XOR(x,y)は x,y のどちらか一方だけが 1 のとき 1 を出力するものですから、
(x,y)＝(0,0),(0,1),(1,0),(1,1)に対して、それぞれ 0,1,1,0 と出力されればＯＫですね。
まずは上の AND 基本素子から見ていきましょう。
上の AND 基本素子には x と NOT(y)が入力されます。
表の赤くなっている部分です。
AND(x,y)は x,y がともに 1 のときに 1 を出力しますから、
(x,NOT(y))＝(1,1)のとき、すなわち(x,y)＝(1,0)のときだけ 1 を出力します。
次は表の緑色になっている部分、下の AND 基本素子について見てみましょう。
NOT(x)と y が入力されるんでした。
上から 2 段目、(NOT(x),y)＝(1,1)のとき、
すなわち(x,y)＝(0,1)のときだけ 1 を出力します。
では、この 2 つの出力が OR に入るとどうなるか。
(x,y)＝(0,0)のとき、(上の AND からの出力,下の AND からの出力)＝(0,0)
(x,y)＝(0,1)のとき、(上の AND からの出力,下の AND からの出力)＝(0,1)
(x,y)＝(1,0)のとき、(上の AND からの出力,下の AND からの出力)＝(1,0)
(x,y)＝(1,1)のとき、(上の AND からの出力,下の AND からの出力)＝(0,0)です。
ここの OR は、上の AND からの出力と下の AND からの出力、

どちらか一方が 1 のとき、1 を出力します。
ですので、最終的には(x,y)＝(0,1),(1,0)のとき、1 が出力されることになります。
(x,y)＝(0,0),(0,1),(1,0),(1,1)に対して、それぞれ 0,1,1,0 と出力されるわけです。
これって XOR(x,y)と同じですよね。
こんな説明でわかるのかしら…

◇EQ、NOR、NAND の置き換え
これは簡単ですね。
否定の関係にあるものを、NOT(　)でガバッとくくってしまえばオッケーです。
NOR(x,y) → NOT(OR(x,y)) みたいにね。

【加算器】

◆２進数の加算の仕方
10 進数と同じ原理 … 各桁ごとに加算 → 結果が 2 以上なら繰り上がり
２進数の下から n 桁目の足し算をすると仮定
足し合わせる２つの数を x、y、
n 桁目に残る数を s、
n＋1 桁目に繰り上がる数を c_{out} とする。
すると図 7-13 のような関係になる。

x	y	s	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

図 7-13

また、n－1 桁目からの繰り上がりを c_{in} とすると、
今度は図 7-14 のような関係になる。

x	y	Cin	s	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

図 7-14

ではこの演算を行う回路はどのようになるか。
まずは図 7-13 の 2 入力 2 出力の回路から考えてみましょう。
x,y と s との関係をよく見てください。
x と y、どちらか一方だけが 1 のとき、s は 1 になってます。
これってどっかで見たことないですか？そうです、XOR です。
つまり、**s＝XOR(x,y)**で表現できるんです。
そして x,y と c_{out} との関係もよくよく見てみてください。
AND の関係が見えます。つまり **c_{out}＝AND(x,y)**です。
これを MIL 記法で表すと図 7-15 のようになり、

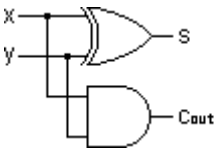

図 7-15

この、下からの繰り上がりが考慮されない加算器を**（1 ビット）半加算器**といいます。
下からの繰り上がりが考慮される加算器を**（1 ビット全加算器）**といい、
２つの半加算器と OR 基本素子を用いて図 7-16 のように構成することが出来ます。

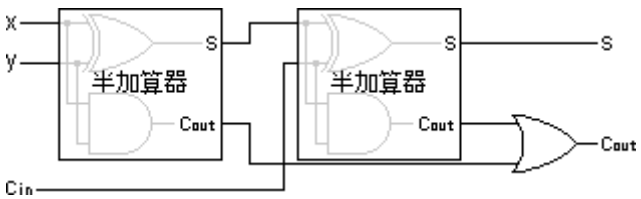

図 7-16

◇**n ビット全加算器**

でも、これではまだ 1011＋1001＝10100 という計算はできません。

このような n ビット（この例では n＝4）の加算をするにはどうするか。

図 7-17 のように、1 ビット全加算器をいっぱい繋げるのです。

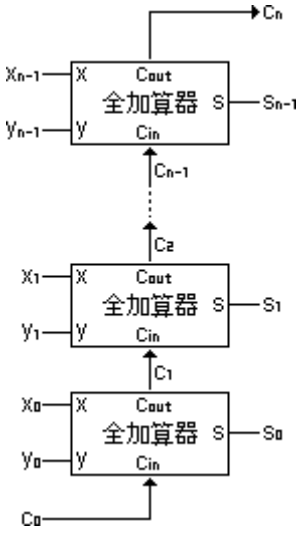

図 7-17

このように、半加算器２つを組み合わせて 1 ビット全加算器を構成し、

1 ビット全加算器を組み合わせて n ビット全加算器を構成する、

という具合に、簡単な回路を組み合わせて複雑な回路を階層的に構成する方法を、**モジュール化**という。

各段階における中間的な構成要素（この場合は半加算器や 1 ビット全加算器）を**モジュール**という。

こうやって簡単勝つ確実な技術を組み合わせることで、

間違いの入り込む余地を減らす作戦らしいのです、はい。

第８章

技術的な話からは抜け出したんで多少楽なんじゃないでしょうか。シケプリもいよいよ後編、ゴールはもうすぐです！みんな頑張ろうじゃないか。

【見えにくい情報システム】

◆情報システムとは

広義 → 情報処理機器（コンピュータなど）＋情報伝達ネットワーク

多様なサービスや機能を提供するシステム

計測・制御系システム、組み込みシステム、ゲーム機なども含む

狭義 → いわゆる**ビジネス・アプリケーション**

… 企業、政府機関、サービス機関などが、

①外部に提供する情報サービスシステム

②内部で用いる、情報管理、意志決定支援システム

→大学でいえば学生証番号・成績の管理など

◆見えない情報システム

ソフトウェアは抽象的記述にすぎず、物理的実体がない

◇組み込みシステム

特定の機能を実現する目的でコンピュータを組み込んでいる、特定目的のシステム

… 携帯電話や炊飯器、車、デジカメなどなど

→ 身近すぎて情報システムとして認識されにくい

◇パソコン … もっともよく目にするコンピュータ

ネットワークの向こう側で見えないコンピュータが多数稼働している

【情報システムの仕組み】

◆チケット予約システムのサービス

◇チケット予約システムの利用者

顧客 … チケットを予約する

主催者 … 公演の日時、席の区分や値段など予約に必要な情報をシステムに与える

システムの管理者 … システムが問題なく稼働し続けるようにシステムを保守

◇顧客から見えるチケット予約システム―提供される機能

▽一般情報照会

・ジャンル別、地域別公演照会

・チケット販売スケジュール照会

▽公演詳細情報照会（顧客が目的の公演を指定すると見られる）

・公演内容（日時、演目）

・公演者（名前、プロファイル）

・公演会場（会場名称、住所）

・チケット情報（座席別料金、座席表）

・予約関連情報（予約受付先、予約受付期間、予約状況）

・主催、協賛（名前、住所）

▽予約機能

・公演の指定

・チケット受け取り方法指定

→ 郵送を指定すると、続いて郵送先住所、受取人氏名、電話番号などの入力

・予約確定

・決済方法指定

→ クレジットカードを指定すると、

続いてカード会社名、カード番号、有効年月、カード所有者名などを入力

▽付随機能

・誤入力の防止、誤入力の処理

Ex.誤入力防止の工夫

16 桁の番号を入力する場合に入力欄を 4 桁ずつ区切る

パスワードの確認入力

・キャンセルの処理

◆チケット予約システムの仕組み

ウェブブラウザからの利用を前提としたサービスが一般的

◇**クライアント／サーバ型**の構成（図 8-1）

サーバ … サービスを提供するプログラム、また、それを搭載したコンピュータ

クライアント … サーバに対してサービスを要求するプログラム

アプリケーション（適用プログラム）

… （この場合はチケット予約に関する）具体的な処理を行うプログラム

Web サーバから呼び出されて処理を行う

データベース … 通常、公演情報や販売状況データが格納される場所

アプリケーションがここにアクセスして情報を得る

▽クライアント―サーバの関係①

Web クライアント（Web ブラウザ）―Web サーバ

▽クライアント―サーバの関係②

アプリケーション―データベース

利用者が直接ウェブサーバにアクセスするのではなく、
電話やファックスでチケット予約センターに連絡をする場合は、
チケット予約センターの受付オペレータが、
利用者に代わってウェブサーバを参照する

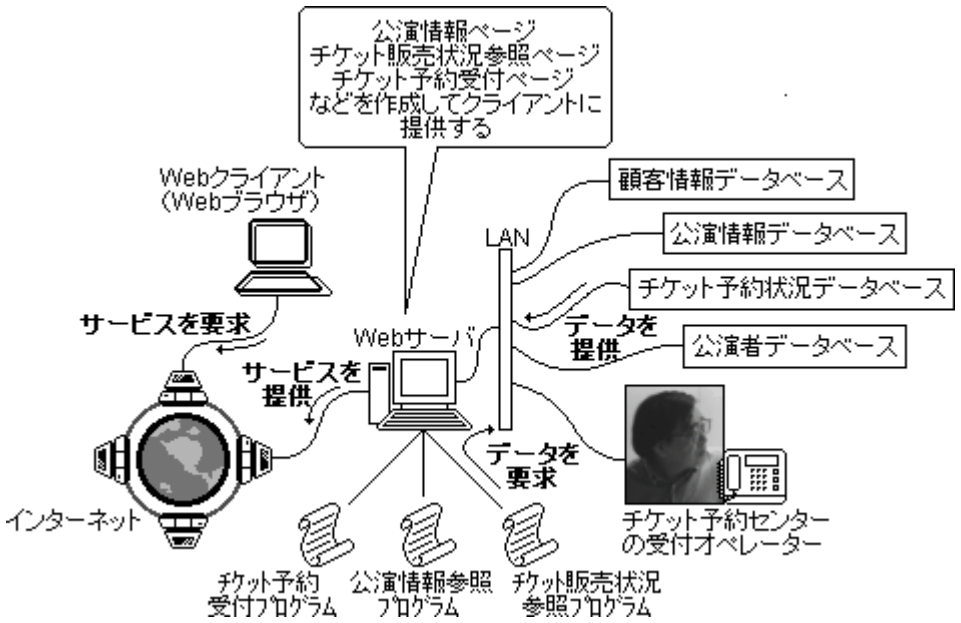

図 8-1

これ全部（写真は除く）自分で書いた！偉くないですか？はい、どうでもいいです。

第９章

世の中、かくも使いにくい物ばかり？と、いうことでね。まぁ、情報機器さんたちからしてみたら「世の中こんなに暮らしやすくしてやってんのに文句言うんじゃねぇ」って感じなのかもしれませんが、使いにくい物は使いにくいです。たぶんこのシケプリも簡潔性がなくて使いにくいと思います。でもあと２章で終わりだよ！頑張ろうじゃないか。

【こんな経験ないですか】

◆暮らしの中の使いにくさ

・ドアを押すのか引くのかわからない
・電灯とスイッチの位置が無関係
・携帯電話の機種によって使い方が違う
・ワープロが勝手に入力を変更する（オートコレクト機能）

◆設計者とユーザの齟齬

Ex.券売機の硬貨投入口

→ コイン詰まりを防ぐための棒が埋め込まれている

→ 「500 円玉は右と左、どちらに入れればよいのですか？」

インタフェース設計者の工夫（device）が予期せぬ形でユーザに解釈されることがある

インタフェースが設計者や人工物（artifact）とユーザとを結ぶ唯一の接点

【インタフェースの定義と機能】

◆定義

インタフェース … もともとは「界面」「接面」など、２つの異なる存在の境界面のこと

本章で扱うのは**ユーザインタフェース、ヒューマンインタフェース**

… コンピュータなどの人工物とユーザ（人間）との間の接面

◆機能

ユーザはインタフェースを通して人工物を操作

→ 人工物が機能を最大限に発揮するためには、使いやすいインタフェースが必要

◇ドライバという（古典的な）人工物

力を伝達したり増幅したりする機能をもつ人工物のひとつ

固いねじには握りの太いドライバ、

細いねじや折れやすいねじには握りの細いドライバが適切

人間の道具（ドライバ）への働きかけが、そのまま対象（ねじ）への働きかけとなる

◇コンピュータという（複雑な）人工物

道具への働きかけが対象への働きかけと等価とは限らない

▽エンターキーを押すという操作

目的は力の伝達や増幅ではなく、情報（意志決定）の伝達

伝達対象は状況によって変わる

→ 漢字変換の確定、ミサイルの発射命令、改行　等

【インタフェースの**二重接面性**】

◆佐伯先生の「第一接面と第二接面」の考え方（図 9-1）

第一接面 … ユーザと人工物との間に存在する直接的なインタフェース

操作インタフェースともいう

第二接面 … 人工物と物理的なタスクとの間に存在する間接的なインタフェース

制御インタフェースともいう

ユーザの目的は第二接面における物理的なタスクの実行

→ 実際に操作できるのは第一接面

ドライバなどの古典的な道具ではほぼ第一接面＝第二接面

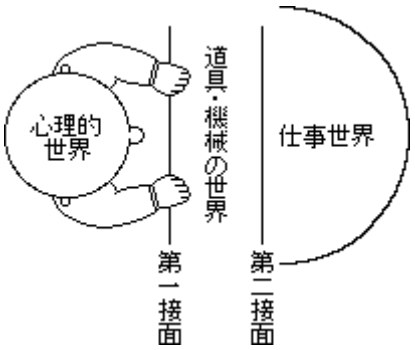

図 9-1

◆コンピュータの汎用性に由来する使いにくさ

◇道具・機械の世界から仕事世界へ（第二接面）の対応づけは多様

・ワープロによる文書の作成

・データベースソフトによるデータの管理

・数値計算プログラムによる建物の構造計算

・ブラウザによるホームページの閲覧　などなど

◇心理的世界から道具・機械の世界へ（第一接面）の対応づけは限定されている

・キーボード

・マウスなどのポインティングデバイス

・ディスプレイや GUI（Graphical User Interface）

第二接面での多様な機能を、第一接面の限られたデバイスで操作可能に

→ エンターキーを押した結果が多岐に渡るという事態は、このような理由による逆向きの対応づけに関しても同じ

→ ソフトウェアの多様な実行結果が、限られた出力デバイスにより提供
（プログラム実行時のエラーメッセージの表示など）

→ ユーザはそれらの情報を元に、自らが操作の善し悪しを解釈する必要

【まとめ】

◆インタフェースのまとめ

◇インタフェースは、システム設計者とユーザとがコミュニケートできる唯一の場

◇システムのユーザビリティ（使いやすさ）の大部分は、
インタフェースのユーザビリティによって左右される

◇インタフェースの構成には、
ユーザの行動特性を考慮し、ユーザ中心の設計思想が求められる

第 10 章

いよいよ最後の章です。いや一長かった。でもこれさえ終われば後はテストを残すのみ！頑張ろうじゃないか。

【技術と社会】

◆技術と社会

情報リテラシー … ただ単にパソコンが操作できるという意味のリテラシーではない
マウスでクリックしたときに“**裏で何が動いているのか**”についておおまかな想像が及ぶ程度のリテラシーを意味する

◇技術と社会を論じる際の４つの論点

①技術の中立性（後述）

②技術と民主主義

③技術倫理

④リテラシー論（後述）

◆情報技術による技術上の変化

◇コンピュータ技術の変遷

▽1960 年代と 1970 年代

巨大データベースの管理と大規模計算

→ 権力の集中、個人のプライバシー保護が論点に

▽1980 年代

マイクロコンピュータの誕生

消費者市場の中にコンピュータ技術が参入

ソフトウェアの所有権に関わる問題が出現

→ ソフトウェアは物理的実体がなく、複製も容易である

人工知能への関心

▽1990 年代以降

インターネット技術の普及

ICT（情報コミュニケーション技術）に関する論考が増えてきている

プライバシーの問題が再燃 → 著作権、匿名性の問題

◇メディアの発展

インターネット技術を含めた情報技術の影響は幅広い

それまでの社会の体制や権力を支えていた構造をひっくり返す力を持つ

社会を変革させる力を持つ … このような論考を**メディア論**という

活版印刷 → 聖書の普及と宗教改革　　映像技術 → 社会主義の崩壊

◇インターネット技術

ホストコンピュータ管理による**閉じられた世界から、**

村（ネットワーク）中心の**開かれた世界へ**（図 10-1）

１つ１つのコンピュータが独立にネットワークに繋がることが出来る技術

→ 相対的に、中央集権的なやり方（ホスト対端末）を弱める

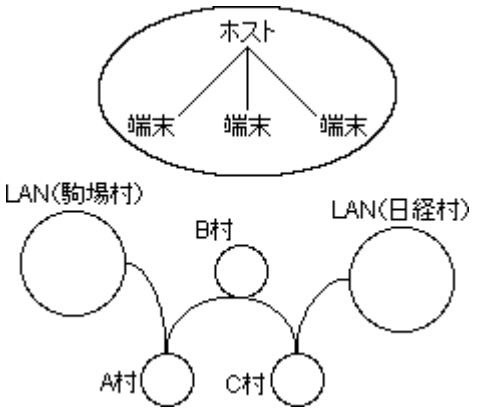

図 10-1

◇４つの変化

①場所の制約からの解放

②時間の制約からの解放

③経路の制約からの解放

④輸送コストをほぼ、ゼロとした

Ex.日本からアメリカへ秒単位でＥメールが送れる

→ ①～④すべて当てはまる

▽以前の活版印刷や映像技術との違い

以前 → ４つの変化は「作り手 → 大衆」という経路に関してのみ当てはまった

今回 → ４つの変化は「１対１レベルのコミュニケーション」の経路でも成立

コミュニケーション形態を一変

… **地理的要素に依存しないコミュニケーション**が可能

→ 国家や地域共同体のような地理的要素に依拠した社会を相対化

【情報技術に固有な社会との軋轢】

◆２つの軋轢

①**無形性**と**複製可能性**（by ジョンソンさん）

… ソフトウェアやディジタルコンテンツは従来の“もの”媒体とは異なる性質を持つ

→ 所有と権利に関わる法制度や倫理との間での軋轢

②**インターネットのグローバルな通信射程**と**匿名性**

→ ネット上での自由で規制のない議論空間、公共空間を構築、電子民主主義の促進

→ プライバシーやセキュリティの軋轢

◇権利と所有概念

従来の“もの媒体” → “電子媒体”

→ 複写する行為や、著作権の侵害に対する利用者の意識の希薄化

▽著作権法

知的財産権 … 知的な創作活動によって何かを作り出した人に対して、
他人に無断で利用されない権利を付与する制度
著作権、工業所有権（このなかに特許権が含まれる）など

> 著作権法第一条
>
> この法律は、著作物並びに実演、レコード、放送及び有線放送に関し著作者の権利及びこれに隣接する権利を定め、これらの文化的所産の公正に留意しつつ、著作者等の権利の保護を図り、もって文化の発展に寄与することを目的とする。

▽著作権法第十条による著作物の例示

1.小説、脚本、論文、講演その他の言語の著作物
2.音楽の著作物
3.舞踏または無言劇の著作物
4.絵画、版画、彫刻その他の美術の著作物
5.建築の著作物
6.地図又は学術的な性質を有する図面、図表、模型その他の図形の著作物
7.映画の著作物
8.写真の著作物
9.プログラムの著作物

◇コンピュータプログラムの著作権

昭和 60 年の著作権法改正でプログラムの定義が明記された

著作物として保護されるもの

… ソースプログラム
オブジェクトプログラム（ソースプログラムをコンパイルしたもの）
オペレーティングシステム（OS）（Windows とか Mac とか）
アプリケーションプログラム（Word とか Internet Explorer とか）

著作物として保護されないもの

… プログラム言語（これ保護されたら新しいプログラム作れません）
規約
解法

▽著作権侵害の事例

事例：A社はビデオゲームXを開発し、その著作権を所有している。B社は都内で経営する喫茶店にゲームXの無断複製ビデオゲームを設置して、顧客に利用させた。A社は著作権侵害を訴えることができるだろうか？

ビデオゲーム機に取り付けられたROMに収納されているオブジェクトプログラムは、A社の著作物（ソースプログラム）の複製物である。したがって、Bが使用したビデオゲーム機のように、ROM のオブジェクトプログラムを他のROMにコピーして製造した偽造ゲーム機は、Aのソースプログラムの著作権を侵害する。

◇**ディジタルコンテンツ**の著作権

使用許諾の概念が曖昧

… メモリ上の電子情報は市場や流通機構なしに直接個人の手に渡ることが可能

→ **DRM（Digital Rights Management）**が問題となる

▽Winny 開発者の基礎問題

Winny … ファイル交換ソフトWinMXの改良版

ファイル交換が完全に匿名化されていることが特徴

→ 匿名性の高さ故に、違法行為を行なってもそのような意識が希薄

多くの音楽や映像、ゲームや他のソフトが著作権者に無断でやりとり

→ ディジタルコンテンツの著作権の侵害を幇助する側面を持つ

アメリカでも似たような事件があった … Napster 裁判

→ 開発者が逮捕されたわけではない

→ 開発者が逮捕されたWinny事件は世界的にも異例な事件

起訴した側の主張

… 現行の法律から鑑みて、

開発者の行動は著作権侵害を幇助するという意味で罪である

開発者の支援者や技術者らの主張

… 技術の進歩とともに法律も進化しなくてはならず、

現在の技術によって簡単に著作権法違反が発生してしまう現状のほうが問題

裁判の結果にかかわらず興味深い事件 … 論点の幅が大きい

→ 技術開発と法、倫理のありかた、市場、文化の保護、自由と規制など、

多くの論点を含んでいる（もっと詳しい解説は教科書 p.242 脚注）

【情報技術論】

◆技術は中立か

技術本質主義 … 技術は、社会の形態や要望にかかわらず
“独立に”発展する、という立場

技術の価値中立論 … 社会への良い面も悪い面も技術の“影響”にすぎず、
技術自体に問題はない、という立場

技術の社会構成主義 … 社会は自身にとって必要な技術をそのつど選択してきたのだ、
という立場
→ 社会の成員による選択と技術とが**共進化**
技術は価値中立でなく、社会の価値により取捨選択される

◇社会構成主義的見方

現在もさまざまな情報技術や情報をめぐる制度が
選択淘汰されつつある時代に生きている、という感覚
→ われわれ一人一人の選択が、次世代の技術に影響
→ 技術開発と同時に社会への影響を考え、選択を公に開くことが必要

◆情報リテラシー

使いやすいインタフェースの設計
→ “裏で何が動いているのか理解しないままにマシンを使う”ユーザの増加

◇情報における**批判的思考（クリティカル・シンキング）**

自分の操作の裏で何が動いているのかについて、ある程度論理的に考える能力
Ex.添付ファイル来た → マウスでクリック、開いてみた → うわぁウィルスだ
→ “マウスでカチッの裏側で何が動いているのか”について最低限の理解の必要
また、情報を主体的に選択、収集、活用、編集、発信するうえでの批判的思考も含む

◇他の技術リテラシーとの違い

情報技術の日常生活への浸透度が高い＋日々の技術革新の速度が速い
→ 広範囲の人に情報における批判的思考の育成が必要
→ 情報技術の特徴（多対多の通信射程、匿名性、複製可能性）を押さえたうえでの
批判的思考が必要

【これからの世代の情報】

ラストです。いよいよみなさんとのお別れが近づいてきました。ところで、ここテストに出るのかな？絶対出ないと思います。でもなんか、藤垣裕子助教授のメッセージがこもってそうなんで、教科書をそのまま抜粋します。って、本当は要約するのが面倒臭かっただけです、ごめんなさい。それではまた会う日まで。31 日かな。See you next time!!!

10.5　これからの世代の情報

これまでの節で概説してきたように、情報技術の普及によって、よい側面（意志決定への直接参加の機会の促進、民主的な公共空間の創出など）とダークサイド（セキュリティとプライバシー、有害情報の規制、既存の社会規範が追いつかないためのさまざまな問題）の双方が同居し、またそれが技術の発展とともに加速されている。同時に、技術が新しい技術にとって代わる、会社が新しい会社にとって代わる、法律が新しい法律にとって代わる、といった社会の新陳代謝も加速されている。

技術と社会規範の新陳代謝が加速されるなかで、これからの若い世代は、面白いものにぶつかるチャンスもあるし、危ないものにぶつかるチャンスも多々あるだろう。その場面場面で技術開発者、利用者、法律の専門家や倫理の専門家などの相互の議論によって解決しなくてはならない問題も多く発生するだろう。しかし、技術の社会構成主義のところでみたように、われわれのいまの選択は、将来世代の情報技術に影響を与えるのである。文明時代の野蛮人になるのではなく、問題が発生するごとにそれらの議論に“参加”し、新しい社会規範の構築に“参加”できるだけの情報リテラシーを身につけて、これからの社会を生き抜いてほしいと思う。