数学入門公開講座

平成3年8月6日(火)から8月15日(木)まで

京都大学数理解析研究所

数学入門公開講座

講師及び内容

1. 整数論・最近の話題（7時間）

京都大学数理解析研究所・教授　伊　原　康　隆

整数論には「フェルマの問題」のように具体的だが人工的な問題が（小石のように）沢山ある一方、はるかに抽象的ではあるが自然な対象も（森、山、雲のように）沢山あります。小石を調べるには岩、山を調べなくてはならないでしょう。具体例をもとに、小石と山の「かかわり」、又研究者の脚力、視力、「登山意慾」といったものについても触れてみたいと思います。

2. パソコンでできる偏微分方程式の数値解法（7時間）

京都大学数理解析研究所・助手　磯　　祐　介

2次元・3次元の微分方程式の問題をパソコンで数値計算することは、メモリーの制限もあって困難な場合が多い。ところが、計算力学の分野で近年よく用いられる〝境界要素法〟は、微分方程式の境界値問題を境界上の関係式に帰着させて数値計算するため、メモリーの節約を図ることができる。

今回は、この境界要素法の解説を行ない、併せて数値解析の理論にも触れる。

3. ナビエ・ストークス流の話（7時間）

京都大学数理解析研究所・助教授　木　田　重　雄

台風や竜巻など何か特別なことが起こらない限りあまり気にならないが、我々をとりまく大気や海洋・河川においては、水や空気がいろいろな形の渦を巻きながら複雑な運動をしている。このような流れを記述するナビエ・ストークス方程式の解の特徴的な振舞を紹介する。

4. 数学とコンピュータ教育（7時間）

京都大学数理解析研究所・助教授　萩　谷　昌　巳

数学者のようなコンピュータの専門家でない人に対するコンピュータ教育について、数学教育と関連させながら考える。

時　間　割

日 / 時間	8月6日(火)	7日(水)	8日(木)	9日(金)	10日(土) 11日(日)	12日(月)	13日(火)	14日(水)	15日(木)
13:15～15:00	伊原	伊原	伊原	伊原	休講	木田	木田	木田	木田
15:00～15:15	休憩					休憩			
15:15～17:00	磯	磯	磯	磯		萩谷	萩谷	萩谷	萩谷

4. 数学とコンピュータ教育（7時間）

京都大学数理解析研究所・助教授　萩　谷　昌　巳

1991, AUGUST 12,13,14,15　　15:15-17:00

数学とコンピュータ教育

萩谷昌己

情報化社会の成熟とともに、コンピュータは社会生活に不可欠なものになりつつあります。それに従って、学校教育の中にも本格的にコンピュータがとり入れられようとしています。しかし、安易なコンピュータ教育は無駄であるばかりでなく有害でさえあります。本講座では、現在の情報処理技術について概観した後、コンピュータに関する教育を分類することにより、コンピュータ教育の本質について考えます。その後で、情報処理技術、特にソフトウェア技術と数学との関連について述べ、ソフトウェア技術に関する教育を既存の数学教育に自然に溶け込ませる方向について模索したいと思っています。

第一回: 情報処理技術概観

第二回: コンピュータ教育

第三回: ソフトウェアの数学

第四回: 数学からソフトウェアへ

● 簡単なプログラミング言語

○ アルゴリズム

計算の具体的な手続きのこと。
必ず停止することを条件とする場合もある。
例：ユークリッドの互除法。素数の判定。

○ プログラミング言語

計算の手続きを記述するための形式的な言語。
記述された計算の手続きは、何らかの方法で、
計算機の上で実行することができる。

○ プログラム

プログラミング言語によって記述された計算の手続き。

○ プログラミング言語の文法

正しいプログラムとは何かを定義する規則。

○ BNF

文法を定義するための方法の一つ。

○ BNFによる文法の定義

--- BNFは、Backus Normal Form、もしくは、Backus-Naur Formの略。

式と項

```
項    ::=   定数
            | 変数
            | ( 式 )
            | 項 * 定数
            | 項 * 変数
            | 項 * ( 式 )

式    ::=   項
            | 式 + 項

定数  ::=   0 | 1 | 2 | ...
変数  ::=   x | y | z | ...
```

--- 「::=」は、左辺が右辺によって定義されることを意味する。
--- 「|」は「または」を意味する。

終端記号

```
* ( ) + 0 1 2 x y z
```

--- *は掛け算。

非終端記号

```
項 式 定数 変数
```

例

```
y+1              式 (項ではない)
x*(y+1)          項 (式でもある)
|
++++++++++       式でも項でもない
+3               式でも項でもない
x-y              式でも項でもない (-は終端記号でない)
```



```
(x+y)*2+3*z          式 (項ではない)
(((x+y)*2)+(3*z))    式 (項でもある)
```

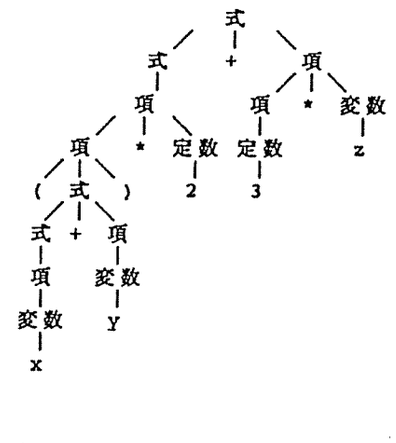
問題

-(引き算)と/(割り算)ができるように上の文法を拡張せよ。

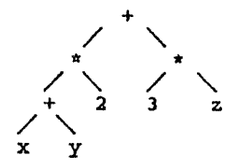
問題

-xのように、マイナスとしての-を許すように、
上の文法を拡張せよ。

○ 導出木



○ 解析木



--- (((x+y)*2)+(3*z))と対応している。

○ 拡張BNF

--- 「::=」と「|」の他に、次のような記号も用いる。

例

```
文      ::=   [ 主語 ] 述語
主語    ::=   名詞 { は | が }
述語    ::=   動詞 | 形容詞 | 形容動詞
```

--- 「[主語]」は、主語があってもなくてもよいことを意味する。
--- 「{は|が}」は、「は」または「が」を意味する。

○ 非常に簡単な、とあるBASICのような言語(TAB)

```
プログラム      ::=
                |     [文番号] 文; プログラム
文              ::=   代入文
                |     IF文
                |     GOTO文
                |     PRINT文
                |     EXIT文
代入文          ::=   LET 変数 = 式
IF文            ::=   IF 式 関係演算子 式 THEN 文
関係演算子      ::=   = | < | <= | > | >= | /=
GOTO文          ::=   GOTO 文番号
PRINT文         ::=   PRINT 式
EXIT文          ::=   EXIT
文番号          ::=   定数
```

○ プログラム1

```
      LET x = 0;
      LET y = 0;
  1   IF x > 10 THEN GOTO 2;
      LET y = y+x;
      LET x = x+1;
      GOTO 1;
  2   PRINT y;
      EXIT;
```

--- xとyの値は次のように変化する。

x	y
0	0
0	0
1	0
1	1
2	1
2	3
3	3
3	6
4	6
4	10
5	10
5	15
6	15
6	21
7	21
7	28
8	28
8	36
9	36
9	45
10	45
10	55
11	55

--- すなわち、EXITしたとき、


```
x = 10
y = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10
```

--- となっている。PRINTされるのは55。

問題

0+1+...+100を求めるように、プログラム1を修正せよ。

○ プログラム2

```
      LET x = 11
      IF x < 3 THEN GOTO 2;
  1   x = x - 3;
      GOTO 1;
  2   PRINT x;
      EXIT;
```

--- xを3で割った余りを計算するプログラム。

問題

商も出力するようにプログラム2を拡張せよ。

○ プログラム3

```
      x = 12
      y = 8
  1   IF x = y THEN GOTO 3;
      IF x < y THEN GOTO 2;
      LET z = x;
      LET x = y;
      LET y = z;
  2   LET z = x;
      LET x = y-x;
      LET y = z;
      GOTO 1;
  3   PRINT x;
      EXIT;
```

--- xとyの最大公約数を求めるプログラム。
--- ユークリッドの互除法の簡単なもの。

○ プログラミング言語の意味論

個々のプログラムが実際にどのような
計算をするのかを定める規則。
文法の各構文の意味を定めることにより、
個々のプログラムの意味が定まる。

例

「PRINT 式」という文は、
式の値を(どこかに)出力する。

--- PRINT文の意味。

--- プログラミング言語の定義は、
--- 文法(syntax)と意味論(semantics)からなる。

● コンピュータのハードウェア

ディジタル回路
コンピュータ・アーキテクチャ

○ 計算機(コンピュータ)の構成

--- コンピュータ・アーキテクチャ

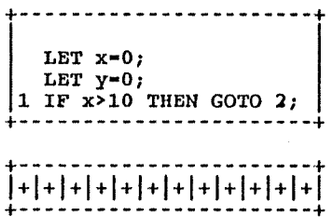
入力装置

出力装置

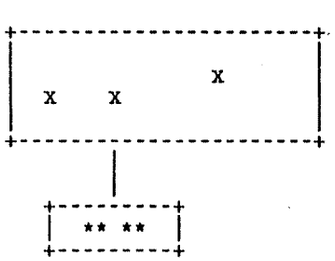
中央処理装置(計算するところ) --- CPU(Central Processing Unit)

記憶装置 --- メモリ、二次記憶装置(ハード・ディスクなど)

パーソナル・コンピュータ



ファミリー・コンピュータ



○ ディジタルとアナログ

ディジタル(digital) --- 整数(0, 1, 2, ..., -1, -2, ...)
アナログ(analog) --- 実数(0.5, 1/3, π, ...)

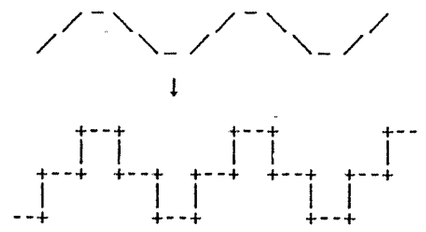
カセット・テープ

音=空気の圧力(アナログ) → 磁気の強さ(アナログ) → 音

コンパクト・ディスク (CD)

音 → 穴の有無(ディジタル) → 音

--- CDのサンプリング周波数は44.8kHz(1Hz=1秒に一回)。
--- ビット数は1サンプリングあたり16ビット(0～65535=2^16-1)。
--- 1時間 = 16*44800*60*60 = 26億



○ ディジタルとアナログの間の変換

AD --- Analog → Digital
DA --- Digital → Analog

○ ディジタル・コンピュータ

ディジタルの形式で計算を行うコンピュータ。
従って、入出力機器がアナログの場合は、
入力に対してはAD変換、出力に対してはDA変換が必要。

キーボード --- ディジタル
紙テープ --- ディジタル
カード --- ディジタル
磁気テープ --- ディジタル
フロッピ・ディスクやハード・ディスク --- ディジタル
プリンタ --- ディジタル
テレビとテレビカメラ --- アナログ
マウス --- アナログ
プロッタ --- アナログ
音声入出力 --- アナログ

○ ディジタル通信

アナログ通信(電話、ラジオ、テレビ)

ディジタル通信(DDX)

○ 入出力装置 --- キーボードとCRTを例に

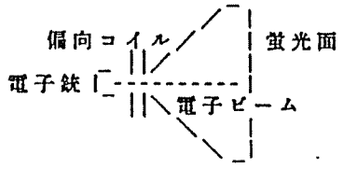
キーボード

だだのスイッチ。

--- 各キーはONかOFF。すなわちディジタル。
--- 本当はもっと複雑。

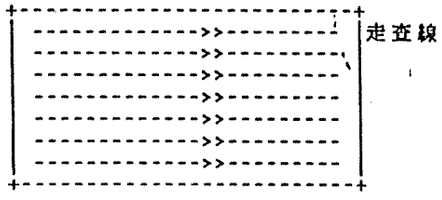
--- 通常は、キーボードがエンコード(符号化)を行い、
--- 文字コードがコンピュータに送られてくる。

CRT --- Cathode Ray Tube (陰極線管)



走査

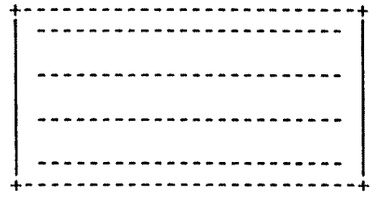
二次元の画像を一次元で送る。

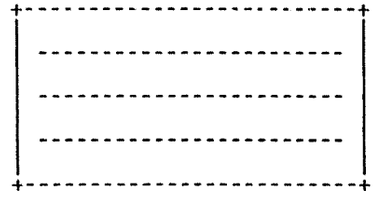


一般のテレビ放送

一画面 = 525本の走査線
一秒 = 30画面

実は2:1のインターレースといって、
奇数番目の走査線と偶数番目の走査線を
交互に表示している。





半分の走査をフィールド、
全走査線の走査をフレームという。

1フレーム = 2フィールド
1フィールド = 525/2本 = 262.5本(263本と262本)
一秒 = 30フレーム = 60フィールド

一本 = 1/30/525 = 63マイクロ秒

--- 実際は見えない部分があるので52マイクロ秒。

CRTへの信号

同期信号と映像信号

--- テレビ放送などでは両者を合わせた複合映像信号を用いる。

--- テレビ放送では、複合映像信号が音声信号とともに、
--- テレビ放送の周波数に変調されて、電波になって飛んでいく。

同期信号

水平同期(走査線の始まりと終り --- 改行)
垂直同期(フレームの始まりと終り --- 改ページ)

映像信号

アナログ --- 各点の輝度 --- 電子ビームの強さ

--- カラーの場合は、RGB(Red、Green、Blue)の輝度が必要(光の三原色)。
--- カラーテレビ放送(NTSC)では、
--- Y=R+G+BとY-BとY-Rが送られる(白黒との互換性のため)。

--- では、どうやってコンピュータに接続され、
--- コンピュータがCRTに文字や絵などを出力することができるのか。
--- → コンピュータ・グラフィックス

○ CPUとメモリ

半導体からなるディジタル回路。

○ ディジタル回路

ディジタル・データを処理する回路。

○ ディジタル・データの表現

電圧の高低による2進数を用いる(正論理)。

電圧の高い(5V) --- 1
電圧の低い(0V) --- 0

--- 実際は多少の余裕がある。
--- 1は2.0V～5.0V。0は0.0V～0.8V。
--- 0.8～2.0Vの値はとってはいけない(壊れた回路)。

--- 1を高い電圧、0を低い電圧で表わすこともある(負論理)。

○ メモリの種類

ROM(Read Only Memory)

読み出し専用メモリ。

RAM(Random Access Memory)

読み書き可能メモリ。

スタティックRAM(Static RAM --- SRAM)

フリップ・フロップによって記憶する。

ダイナミックRAM(Dynamic RAM --- DRAM)

コンデンサの容量によって記憶する。
時間が経つとコンデンサが放電するので記憶が失われる。
このために、一定時間ごとに、
「メモリの内容を読んでそのまま書く」
という操作を行う必要がある。
これをリフレッシュ(refresh)という。

--- 1メガDRAM＝1個のチップに1Mbit記憶できる。
--- これが標準になりつつある。
--- 1文字＝16bitとすると、1個のチップで約6万字。

○ CPU

命令デコーダ --- 命令の解釈と実行
ALU(Arithmetic&Logic Unit) --- 算術論理演算
命令レジスタ --- 実行中の命令を保持しておくレジスタ
プログラム・カウンタ --- 次の命令のアドレスを保持しておくレジスタ
汎用レジスタ --- 汎用のレジスタ(算術論理演算等に用いる)

--- レジスタ(register)とはCPUの中にある高速メモリ。

○ 計算機(ハードウェア)の種類とその規模

	速度	メモリ	ディスク
スーパー・コンピュータ			
汎用大型機			
ミニ・コンピュータ	1-6 MIPS	1-32 MB	
ワークステーション(5年前)	1-4 MIPS	4-16 MB	
ワークステーション(今)	10-20 MIPS	4-16 MB	
パーソナル・コンピュータ	0.1-1 MIPS	256 KB - 8 MB	--20MB

北白川にあるとある研究所では...

DG/MV10000	2.5 MIPS	12 MB	1 GB
ENCORE/Multimax	2 MIPS x 8	80 MB	800 MB
SUN2	0.5 MIPS	4MB	
SUN3	1 MIPS	8 MB	150 MB
SONY/NEWS	1 MIPS	4 MB	150 MB
SUN4/260	10 MIPS		
SUN/SparcStation1	12 MIPS	8 MB	200 MB
SUN/SparcStation2	28.5 MIPS	16 MB	400 MB
SUN/SparcServer	20 MIPS	64 MB	6 GB
Apollo/DN3500	4 MIPS	8 MB	
Apollo/DN4500	7 MIPS	8 MB	
Apollo/DN10000	20 MIPS		
MacSE/30	3 MIPS	8 MB	
MacII	3 MIPS	8 MB	
MacIIci	4 MIPS	8 MB	

--- MIPS = Mega Instructions Per Second
--- GFLOPS = Giga FLoating Operations Per Second
--- MB = Mega Byte
--- GB = Giga Byte

○ 数の表現について

2の補数表現

例(4bit)

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

n bitの2の補数表現

x(n-1) x(n-2) ... x(2) x(1) x(0) (x(i) = 0 または 1)

は、

- x(n-1)*2^n-1 + x(n-2)*2^n-2 + ... + x(2)*2^2 + x(1)*2 + x(0)

を表わす。
例えば、4bitの1011は、

-1*2^3 + 0*2^2 + 1*2 + 1 = -8 + 0 + 2 + 1 = -5

となる。
2の補数表現の利点は、
正の数のためのadderをそのまま負の数にも使えること。
n bitの補数表現で表現できる数の範囲は、

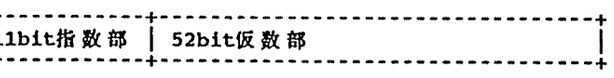
- 2^n-1 -- 2^n-1 - 1

である。

8bit	-128 -- 127
16bit	-32768 -- 32767
32bit	-2147483648 -- 2147483647

浮動小数点数

浮動小数点数の表現には色々な方法があるが、
どれも指数部(exponent)と仮数部(mantissa)とから成る。
ここでは、64bitのIEEE形式を考える。



符号bit(1bit) --- 0は正、1は負。
11bit指数部 --- 2進数(0--2047)。
52bit仮数部 --- 「0.1仮数部」という2進小数を表わす。

指数部の表わす整数をe、仮数部の表わす小数をmとすると、
指数部と仮数部で、

```
m * 2^(e-1022)
```

という小数を表わす。これに符号bitが加味される。
0は例外で、64bitをすべて0にしたもので表現する。

例

```
1 10000000000 0100000000000000000000000000000000000000000000000000

0.101(2) = 1/2 + 1/8 = 0.5 + 0.125 = 0.625

-0.625*2^(1024-1022) = -0.625*4 = -2.5
```


● 言語処理系

○ とあるマシン(TAM)

--- 教育用の仮想的なマシン(おもちゃ)。

プログラム・カウンタ --- 8bit

汎用レジスタ --- 8bit
r0--r15(16個)

アドレス・パス --- 8bit
データ・パス --- 8bit

命令

0000 rrrr aaaaaaaa	load	メモリ→レジスタ
0001 rrrr dddddddd	load immediate	データ→レジスタ
0010 rrrr aaaaaaaa	store	レジスタ→メモリ
0011 rrrr rrrr rrrr	add	加
0100 rrrr rrrr rrrr	sub	減
0101 rrrr rrrr rrrr	mul	乗
0110 rrrr rrrr rrrr	div	除
01110000 aaaaaaaa	jump	ジャンプ
01110001 rrrr rrrr aaaaaaaa	jeq	等しければジャンプ
01110010 rrrr rrrr aaaaaaaa	jneq	等しくなければジャンプ
01110011 rrrr rrrr aaaaaaaa	jlt	小さければジャンプ
01110100 rrrr rrrr aaaaaaaa	jleq	≦ならばジャンプ
01110101 rrrr rrrr aaaaaaaa	jgt	大きければジャンプ
01110110 rrrr rrrr aaaaaaaa	jgeq	≧ならばジャンプ
10000000	halt	停止
1001 rrrr	input	入力
1010 rrrr	output	出力

--- aaaaaaaaはアドレスの指定。
--- ddddddddはデータを直接に指定。
--- rrrrはレジスタの指定。

拡張命令

10000001 rrrr rrrr	move	レジスタ→レジスタ
10000010 rrrr rrrr	load deferred	間接load
10000011 rrrr rrrr aaaaaaaa	load indexed	インデックスload
10000100 rrrr rrrr	store deferred	間接store
10000101 rrrr rrrr aaaaaaaa	store indexed	インデックスstore

例

```
0000 0010 00000101              5番地の内容をr2へ入れる。

0000            load            オペコード(opcode)
0010            r2              オペランド(operand)
000000101       5番地           オペランド(operand)

0001 0010 00000101              5をr2へ入れる。

0001            load immediate
0010            r2
000000101       5

0010 0010 00000101              r2の内容を5番地に入れる。
```

```
0010            store
0010            r2
00000101        5番地

0101 0010 0011 0100             r2*r3の値をr4に入れる。

0101            mul
0010            r2
0011            r3
0100            r4

01110000 00001111               15番地にジャンプする。

01110000        jump
00001111        15番地

01110100 0000 0001 00001111     r0≦r1ならば15番地にジャンプする。

01110100        jleq (jump less or equal)
0000            r0
0001            r1
00001111        15番地

10000000                        停止する。

10000000        halt

1010 0010                       r2の内容を出力する。

1010            output
0010            r2
```

○ アセンブリ言語

機械語のオペコードやオペランドを、
2進法ではなく人間にわかりやすい記号で書けるようにした言語。
特にオペコードを表わす記号をニーモニック(mnemomic)という。

--- load、store等がニーモニック。

マシンの一つ一つに対してそのマシン専用のアセンブリ言語がある。
アセンブリ言語で書かれたプログラム(アセンブリ・プログラム)を
機械語に翻訳することを「アセンブルする」という。
アセンブルするもの(処理系)をアセンブラという。
アセンブラも一つのプログラムであり、
何らかのプログラミング言語で書かれている。
(アセンブラ自身で書かれているかもしれないし、
別の言語で書かれているかもしれない。)

とあるアセンブリ言語(TAA)

--- TAM用のアセンブリ言語

例

```
load r2,5
load r2,#5
store r2,5
mul r2,r3,r4
jump 15
jleq r0,r1,15
halt
output r2
```

--- load r2,#5は、load immediateにアセンブルされる。

ラベル

アセンブリ・プログラムの中では、
x,y,L1,L2のようなラベル(label)を用いることができる。

```
load r2,x
jump L1
jleq r0,r1,L1
```

アセンブラはラベルを実際のアドレスで置き換える。

○ コンパイル

FORTRANやBASICのような高級言語を、
アセンブリ言語もしくは機械語に翻訳すること。

とあるBASICのような言語(TAB) --> TAA または TAM

例

TAB

```
   LET x = 0;
   LET y = 0;
1  IF x > 10 THEN GOTO 2;
   LET y = y+x;
   LET x = x+1;
   GOTO 1;
2  PRINT y;
   EXIT;
```

TAA

```
        load r0,#0
        store r0,x
        load r0,#0
        store r0,y
L1:     load r0,x
        load r1,#10
        jgt r0,r1,L2
        load r0,y
        load r1,x
        add r0,r1,r0
        store r0,y
        load r0,x
        load r1,#1
        add r0,r1,r0
        store r0,x
        jump L1
L2:     load r0,y
        output r0
        halt

x:      0
y:      0
```

TAM

```
番地(10進)
  0     0001 0000 00000000
  2     0010 0000 01000000
  4     0001 0000 00000000
  6     0010 0000 01000001
  8     0000 0000 01000000
 10     0001 0001 00001010
 12     0111 0101 0000 0001 00100001
 15     0000 0000 01000001
 17     0000 0001 01000000
 19     0011 0000 0001 0000
 21     0010 0000 01000001
 23     0000 0000 01000000
 25     0001 0001 00000001
 27     0011 0000 0001 0000
 29     0010 0000 01000000
 31     01110000 00001000
 33     0000 0000 01000001
 35     1010 0000
 36     10000000

 64     00000000
 65     00000000
```

--- xとyの初期値を0としているので、
--- 最初の4命令は必要ない。
--- 機械語は適当に区切ってあるが、
--- これは単に読み易さのためである。

問題

TABの他のプログラムをTAAにコンパイルし、

さらにTAMにアセンブルしてみよ。

○ 高級言語

目的に応じて様々な高級言語がある。
コンパイラやインタープリタによって実行される。
マシンに依存せず同じプログラムを異るマシンで実行することができる。

--- 各アプリケーションが専用の言語を持っていることがある。
--- 例えば、データ・ベース、リンカ、エディタ、ウィンドウ・システム。

○ コンパイラとインタープリタ

TABのプログラムを人間が実行することもできる。
それと同じことをプログラムの形にして
計算機に実行させることもできるはず。
そのプログラムがインタープリタである。
例えばBASICはBASICインタープリタによって実行される。

コンパイラのように
プログラム全体を処理する必要がないので応答が速く、
プログラムそのものを実行するので、
プログラムの誤りの修正等が容易。

ただし、プログラムの実行は、
コンパイラによって機械語に翻訳してから実行する方が速い。

○ コンパイラ

コンパイラの入力となる高級言語で書かれたプログラムを、
ソース・プログラム、ソース・コード、ソース・モジュールという。
コンパイラの出力である機械語のプログラムを、
オブジェクト・プログラム、オブジェクト・コード、
オブジェクト・モジュールという。

--- モジュールとはプログラムの断片のこと。

コンパイラの構成要素

構文解釈	プログラムが文法に適合しているかを調べながら、
＋	オプティマイズやコード生成がやりやすいような
意味解析	データ構造(例えば木構造)を生成する。
オプティマイザ	オブジェクト・コードの効率を上げる(オプティマイズする)ために様々な処理を行う。
コード生成	コードの生成(アセンブリ言語、機械語、中間言語)。

○ リンカ・ローダ

オブジェクト・コードには、
メモリの中の定まったのアドレスでしか実行できないものと、
どのようなアドレスに置いても実行できるものがある。
後者をリロケータブルなオブジェクト・コードという。
前者を(ロードがすんでいるいるという意味で)ロード・モジュールという。

リロケータブルなオブジェクト・コードを、
与えられたアドレスで実行できるようにすることを、
リロケーション、もしくは、ローディングという。
ローディングを行う処理系をローダという。

また、いくつかのリロケータブル・プログラムを結合して、
一つのリロケータブル・プログラム、もしくは、
ロード・モジュールを生成することをリンクするという。
リンクを行う処理系をリンカという。

○ ライブラリ

プログラムを集めて一まとめにしたもの。
ユーザが自分で書いたプログラムを集めて
ライブラリにすることもできるが、
コンパイラ等の言語処理系の一部として、
最初から便利な機能がライブラリの形で提供されている。

例

数値計算のライブラリ(SIN,COS,GAMMA)
グラフィックスのライブラリ

ライブラリを用いている(呼び出す)プログラムとライブラリを
結合するのもリンカの仕事。

○ プログラミング言語の三つの「構造」

データ構造(data structure)

どのような種類の構造のデータが用意されているか。

配列
レコード
ポインタ
リスト
スタック

制御構造(control structure)

プログラムの実行を制御するために、
どのような種類の命令が用意されているか。

分岐(IF)
繰り返し(WHILE,REPEAT,FOR)
ジャンプ(GOTO)
再帰呼び出し

プログラム構造(program structure)

プログラムがどのような構成要素からなり、
それらを組み合わせるためにどのような機構が用意されているか。

手続き定義
関数定義
ブロック構造
モジュール構造

○ プログラミング言語の例

FORTRAN(フォートラン)
数値計算用の言語。
1950年代後半にIBMで開発された。
現在では1977年に制定されたFORTRAN77と呼ばれる方言が
よく用いられている。

BASIC(ベーシック)
FORTRANを簡略化した言語。
1964年にダートマス大学で開発された。
パソコンの標準言語となっている。
インタープリタで対話的に使用できる。

COBOL(コボル)
事務処理用言語。
1960年にCODASYL(データ・システム言語協議会)が制定。
(量的に)最も多く使われている言語。

PL/I(ピーエル・ワン)
FORTRANとCOBOLにAlgol的な制御構造とデータ構造を加味した汎用言語。
IBMの標準語として開発された。

APL(エーピーエル)
BASICと同様の対話型言語。
行列の計算等を得意とする。
IBMが標準言語として採用している。

Algol(アルゴル)
ヨーロッパで開発された汎用の言語。
整理されたデータ構造、制御構造、プログラム構造を持つ。
初期のものをAlgol60といい、
後に新たに設計され直されたものをAlgol68という。

Pascal(パスカル)
1968年にチューリッヒ工科大学のヴィルトが開発した言語。
Algol60を簡略化・効率化した言語。

Modula-2(モジュラ・ツー)
ヴィルトがPascalにモジュール構造を導入して開発した言語。

C(シー)
AT&Tのベル研究所で開発されたシステム記述用の言語。
UNIXと呼ばれるオペレーティング・システムの記述に用いられた。

C++(シー・プラス・プラス)
C言語を拡張しオブジェクト指向言語にしたもの。

ADA(エイダ)
DoD(米国国防総省)が公募し、
フランスのHoneywell Bullが開発した汎用言語。
軍事兵器の制御等に用いられる。

LISP(リスプ)
記号処理(数式処理・人工知能)のための言語。
1950年代後半にMITのマッカーシーらによって開発された。
インタープリタで対話的に使用できる。
現在はCommon Lispと呼ぶ方言が標準になりつつある。

Scheme(スキーム)
LISPを理論的な観点から洗練した言語。

Prolog(プロローグ)
論理型言語。人工知能に用いられる。
日本の第五世代コンピュータ・プロジェクトで採用された。

Smalltalk-80(スモールトーク・エイティ)
オブジェクト指向型言語。
単なるプログラミング言語というよりも、
オブジェクト指向の原理によって構築された
一つのコンピュータ・システム。

LOGO(ロゴ)
MITのパパートが開発した幼児教育のための言語。
ロボット幾何学が特徴。

● オペレーティング・システム(Operating System、OS)

○ 広義のオペレーティング・システム

アプリケーション・ソフトウェアを走らせるために、
縁の下の力持ちとなって働くソフトウェア。
基本ソフトウェアという。

コンパイラ、アセンブラ等の言語処理系も
広義のオペレーティング・システムには含まれる。

オペレーティング・システムは、
計算機の一部として売られ、
計算機のメーカが提供する(例外もある---重要)。

○ 狭義のオペレーティング・システム

基本的な入出力の機能を提供し、
プロセスやファイルを管理するソフトウェア。
カーネル(核)ともいう。

入出力管理
プロセス管理
ファイル管理

オペレーティング・システムの他のソフトウェアや、
アプリケーション・ソフトウェアは、
ハードウェアを直接に操作せずに、
カーネルに仕事を依頼する(システム・コール)。

高級言語で書かれたプログラムが
複数のCPU上で実行できるように、
ハードウェアを直接に操作せずに、
システム・コールのみを用いるプログラムは、
同じOSが稼働する複数のハードウェア上で実行できる。
--- CPUが同じでも入出力のハードウェアが異なれば入出力の方法が異なる。

○ 身近なオペレーティング・システム

MS-DOSは、NEC、富士通、IBMなどのパソコン上で稼働する。
CPUは同じだが周辺ハードウェアは異なる。
MS-DOSのシステム・コールのみを用いたプログラムは、
どのメーカのパソコンでも実行できる。

UNIXは異なるCPU上で稼働する。
CPUも周辺ハードウェアも異なる。
UNIX上の高級言語(例えばC)で書かれたプログラムは、
UNIXの稼働する計算機上で実行できる

IBMコンパチブル(コンパチ)とは、
CPUも周辺ハードウェアもIBMに同じ。
オペレーティング・システムもIBMに同じ。

○ 入出力管理

例えば端末への文字の表示。
ユーザのプログラムがハードウェアを直接に操作して、
文字を端末に出力するのは大変であり、
ハードウェアに依存したプログラムになってしまう。

入出力管理は、基本的な入出力機能を、
システム・コールの形で提供する。

○ プロセス管理

実行中のプログラムをプロセスまたはタスクという。

CPUは一つでも同時に二つ以上のプログラムを走らせることができる。
一つのプロセスが入出力待ちで止まっている間、
他のプロセスを実行できる。

CPUを時分割してプロセスに割り振ることもできる(TSS)。

これを、マルチ・プログラミング、
マルチ・プロセシング、
マルチ・タスキングという。

○ ファイル管理

データの集合をファイルという。
通常は二次記憶上に作られる。

ファイルはファイル名で参照される。

ファイル管理の仕事は、
ファイル名と実体の対応、
ファイルの実体の割り付け、
ファイル名の管理、
ファイルの保護。

○ オペレーティング・システムの例

OS360 --- IBM
Multix
Tenex --- DEC
UNIX
CP/M
MS-DOS

○ 広義のオペレーティング・システムの構成要素

カーネルから見るとアプリケーション・ソフトウェアと同等。

言語処理系

ソフトウェアの開発のための処理系

エディタ
マクロ・プロセッサ
ライブラリ管理
バージョン管理
フォーマッタ(清書系)
ウィンドウ・システム

日本語入力

通信(ネットワーク)

○ オペレーティング・システムの種類

バッチ

一まとまりの仕事(ジョブ)をカードにパンチして
カード・リーダから入力する。
結果はプリンタから出力される。
OSは順にジョブを実行する。

TSS(Time Sharing System)

端末から計算機を利用する。
各プロセスに時分割でCPU時間を割り当てる。

各ユーザが計算機を独占しているように見える。
すぐに結果が見れる。
ユーザが多くなると効率が大幅に落ちる(スラッシング)。

実時間

時間的制約があるTSS。
例えば工場の制御。

分散

ワークステーションと呼ばれる計算機を
ネットワークで結合して用いる。
自分の計算機があり、しかも、
自分の計算機の資源と他の計算機の資源を共有することもできる。

教養としてのコンピュータ教育

以下では、教養としてのコンピュータ教育を、

* 文房具教育
* 実験器具教育
* 語学教育
* 計算機導入演習
* 計算機概説
* 計算基礎論入門

の六つに分けて考察してみたい。これらの教育は、その中身もその目的も全く違っているものである。従って、その区別をはっきりさせることからして重要なのである。

* 文房具教育

さて、教養としてのコンピュータ教育のうちで、今後ますますとして重要になってくるものが、文房具としてのコンピュータの教育である。

文房具としてのコンピュータの用途には、次のようなものがある。

+ 文書処理
+ コミュニケーション
+ プレゼンテーション

+ 文書処理

文書処理の代表はワープロである。ワープロは、編集系(テキスト・エディタ)と整形系(フォーマッタ)が融合したシステムであり、できあがりの文書のイメージを直接に編集することができる。文書には図や絵も含まれるから、グラフィック・エディタもワープロの一部と考えた方がよい。なお、ワープロを高機能化して本の組版をも行えるようにしたものがDTP(Desk Top Publishing)システムである。

ワープロは、大きく、ワープロ専用機と、パソコンやワークステーション上のワープロ・ソフトの二つに分けることができる。いうまでもなく、他のソフトウェアと文書のやり取りができるという点において、無限の可能性を秘めているのは後者である。もちろん、ワープロ専用機でも通信機能がしっかりしていれば、パソコンやワークステーション上のソフトウェアと文書のやり取りをすることができるだろう。

ワープロ以外の文書処理には次のようなものがある。

スケジュール管理
アドレス帳
辞書

これらの機能は従来は紙の手帳によって実現されていたものであるが、いまや、電子手帳という文明の利器に凝縮されているようである。また、これらの機能は、MacintoshのHyperCardの機能でもある。従って、文房具教育においてHyperCardを教えるのは極めて妥当なことかも知れない。

ワークステーションとは、一言でいうと、計算速度、記憶容量、ネットワーク機能などにおいて、最も優れた個人用のコンピュータのことである。そのワークステーションも、最近では、ますます文房具として使われることが多くなって来た。計算能力を理由にワークステーションを選ぶこともあるが、ワークステーションを使う大きな理由は、やはり、ネットワークにあると思う。ワークステーションのネットワーク機能を使いたいがために、文房具全体をワークステーション上で走らせることが多い。

ワークステーション上では、ワープロの機能は汎用のテキスト・エディタと独立したフォーマッタによって実現されている。また、電子手帳の個々の機能は、独立したアプリケーションによるか、テキスト・エディタで工夫して行う。

汎用のテキスト・エディタとしては、いまとなっては、GNU Emacs以外には考

えられない。GNU Emacsは、その拡張機能によって、単なるテキスト・エディタとしてではなく、後述する電子メールや電子ニュースを始めとする様々なアプリケーションのユーザ・インターフェースとしても用いられている。

フォーマッタとしては、いまとなっては、LaTeX以外には考えられない。なお、スタイル・ファイルの書き方まで教養で教えるべきかどうかは議論の余地がある。

GNU Emacs、TeXのプレビュアー、グラフィック・エディタなどをワークステーション上で動かすためには、ウィンドウ・システムが必要である。ウィンドウ・システムとしては、いまとなっては、X11以外には考えられない。

そして、ワークステーション上で以上の処理系を使うためには、フィイル・システムなどのオペレーティング・システムの基礎知識がどうしても必要である。従って、文房具教育といえども、ワークステーションを使う場合は、オペレーティング・システムについて教えないわけにはいかない。ワークステーションのオペレーティング・システムとしては、いまとなっては、UNIX以外には考えられない。

さて、日本語の文書を扱うためには、ワープロ専用機であろうとワークステーションであろうと、何らかの方法で日本語を入力しなくてはならない。日本語入力法としては、いまとなっては、仮名漢字変換以外には考えられない。特に、逐文節変換もしくは先読み変換が常識である。さいわい、ワークステーション上にも、すぐれた仮名漢字変換システムが多く開発されている。

以上、文書処理について簡単に説明したが、いまなおタイピングの練習は重要である。特に、タッチ・タイプの修得は人生の財産であるといっても過言ではない。従って、難しいワープロやHyperCardを覚えるよりも、少しでもタイピングの練習をした方がよいのかもしれない。

+ コミュニケーション

コンピュータが従来の文房具を越えることができる最大の理由はコミュニケーションにあるといってもけっして過言ではない。コミュニケーションの機能としては、大きく、電子メールと電子ニュースがある。

電子メールは、広域ネットワークを通して自分の計算機から相手の計算機に手紙を送る機能である。手紙は通常はテキストのみから成るが、図や絵の入った文書を送れるような電子メール・システムもある。

電子ニュースは電子掲示板とも言われる。特定のトピックスに関して不特定多数の人と意見を交換するための機能であるが、会議の案内を始めとする事務連絡にも使われる。従って、将来的には、電子ニュースが読めなくては、組織人として無能になってしまう可能性さえある。

なお、コミュニケーションを行うためには、先にも述べたように、どうしてもワークステーションがよい。特に、SUNなどの標準的なワークステーションがよい。これは、最先端の機能を最小の労力で得ることができるためである。

+ プレゼンテーション

ここでいうプレゼンテーションとは、グラフィックスやアニメーションを用いて、研究の発表、商品の宣伝、企画の提示などを行うことをいう。Macintoshなどには、この手のソフトが色々とあるようである。

+ プログラミングは必要か

最後に、文房具教育においてプログラミングは必要か、ということについて考えてみたい。

GNU EmacsやHyperCardは、文房具といえども、その機能を拡張するためのプログラミング言語を持ったプログラミング・システムでもある。これらのシステムをフルに使うためには、どうしてもプログラムを書く必要が生じる。しかし、既に他人が用意してくれているライブラリ(HyperCardではスタック)が数多く存在するので、よほど特殊なことをしようとしない限り、そのようなライブラリを使えばことは足りる。従って、純粋に文房具として使うためには、プログラ

ラミングは必要ない。

文房具教育においてプログラミングを禁止する必要はない。人によってはやってもよい。しかし、必修にすべきではないだろう。

* 実験器具教育

従来から大学の教養課程で行われていたコンピュータ教育は、ここでいう実験器具教育であったと考えられる。実験器具教育とは、実験を行うための道具の使い方を教えることであり、それ自体が大学教育の目標ではない。

例えば、大学の教養課程の化学実験でも極めて精密な秤を用いるが、その使い方は非常に難しい。そこで、化学実験に先立って秤の使い方の講習を受けることになる。これは、まさに、実験器具教育ということができよう。その他、顕微鏡の使い方や、レベルは低いが、試験管の洗い方などが、実験器具教育である。

かつては、電卓も物理学実験や化学実験の道具であった。従って、電卓の使い方を教えることも実験器具教育である。実験器具としてのコンピュータは、電卓を高機能にしたものということができる。コンピュータともなると、物理学実験や化学実験を始める前のちょっとした時間だけでは、とてもその使い方を教えることはできない。従って、コンピュータという実験器具の教育には、実験の授業とは独立した授業がどうしても必要になる。

実験器具としてのコンピュータの用途は、次のように三つに分けられる。

* データの記憶
* データの処理
* データの表示

+ データの記憶

卑しくも実験というからには何らかの観測データが得られるはずである。そして、卑しくもコンピュータというものは記憶装置を持っているのであるから、そのような観測データを記憶することができる。特に、データが大量で複雑な構造を持っているとき、コンピュータはその威力を発揮する。

パソコンやワークステーションでデータを記憶するために最も簡単な方法は、普通のファイルを用いることである。形式は単純なテキストにすればよい。しかし、卑しくもファイルというものを使うには、ファイル・システムというものを知らなくてはならない。従って、ディレクトリなどのファイル・システムの概念を教える必要が生じる。

単なる数値の並びではなく、構造を持った情報を記憶する手段としてデータベースがある。データベースを使えば、データの記憶だけでなく、データの検索とデータの簡単な処理を行うこともできる。

データベースの教育の対象は、現在のところは、関係データベースであろう。しかし、やがては、オブジェクト指向データベースも教養になるかもしれない(そう期待したい)。また、データが大量になると、一つの機関で入力したり記憶したりすることができなくなるので、実験器具教育でも、遅かれ早かれ、分散データベースを教えるようになるだろう。

+ データの処理

従来から行われているFortranやBasicの教育は、実験結果の処理を行うためのものである。平均や分散などを求める単純な統計処理に始まり、実験結果をフーリエ変換したり、χ^2検定のような高度な統計処理をしたりするために、FortranやBasicのプログラムを書くのが常であった。

近年では、簡単な統計処理ならば、スプレッド・シートを始めとする簡易ソフトを用いることにより、プログラムを全く書かなくてもできるようになっている。Fortranなどでも、昔から、既存のライブラリをそのまま使うことが多かった。統計用のパッケージであるSPSSは、その典型的な例である。

いわゆるシミュレーションのための計算は、実験結果の処理というよりも、実

験そのものである。従って、計算機は風洞や原子炉のような実験設備と同等のものであるということができる。従来より、シミュレーションのための言語はFortranであり、Fortranを用いて、行列計算、微積分、有限要素法などの数値計算を行うプログラムが書かれて来た。また、多くのライブラリがこのような目的のために用意されている。

数学の研究における数式処理システムの利用は、シミュレーションによる実験にかなり近いものである。従って、教養の実験器具教育として、ReduceやMacsymaなどの数式処理システムを教えることは極めて重要である。ちなみに、最近では、Mathematicaという商業ベースの数式処理システムも登場した。

+ データの表示

実験結果をグラフやダイアグラムで表示することは、実験結果から新たな知見を得るために非常に有用である。従って、遠い昔から、XYプロッターなどを用いてデータを表示することが行われてきた。このような中から、GKSやCOREなどのグラフィックス・パッケージが生まれたといっても過言ではないだろう。

最近では、簡易ソフトにグラフィックスの機能が用意されていて、データの記憶・処理から表示に至るまでを、プログラムを全く書かなくても行えるようになり始めている。また、Mathematicaのような数式処理システムでも、グラフィックスに力をいれるようになった。

+ プログラミングは必要か

では、実験器具としてのコンピュータのために、将来的には、プログラミングは必要なくなるのであろうか。これはなかなか難しい問題である。簡単な計算に関しては簡易ソフトで十分にカバーできるだろうが、より複雑なデータの処理や表示に対してはそうもいかない。しかし、需要の大きな分野においては、問題ごとに特化したソフトウェアがどんどんと作られていく可能性は大きい。現実に、化学などの分野では、そのようなソフトが随分と見受けられるようになった。そうなって来ると、普通の人々は、ほとんどプログラムを書く必要がなくなるだろう。従って、実験器具教育でも、プログラミングを教える必要はなくなっていく。

* 語学教育

第二外国語は教養課程における最も典型的な科目である。なぜ第二外国語を教養で教えるのだろうか。これほど英語が国際化してしまった現在、第二外国語を教える意義などあるのだろうか。特に理科系において。

第二外国語は役に立たないという前提に立ったとき、第二外国語教育の意義を無理やり付けるとすると、次のようになる。

+ 日本語と英語以外の言語の体験
+ 文法の修得
+ 文章力の向上
+ 暗記力の鍛錬
+ 忍耐力の修行

文房具教育においても、実験器具教育においても、プログラミングの必要性はますますなくなっていく。とすると、プログラミング教育も、ますます、第二外国語教育に近いものになっていくように思われる。実際に、プログラミング教育も上のようにして意義付けせざるを得ないのである。以下、これを検証してみよう。

+ 日本語と英語以外の言語の体験

コンピュータは、プログラミング言語と呼ばれる言語で書かれたプログラムというものによって動く。従って、たとえ一生の間にプログラムを書く必要など全くない普通の人であっても、プログラムを書くことがどのようなことであるかを一度体験してみるのは教養として非常によいことである。

+ 文法の修得

第二外国語の教育、特に、文法の説明は、英語に比べて非常に短期間で行われ

る。ドイツ語やフランス語など、半年間で全文法の説明を終ってしまう。従って、第二外国語を習うと、非常に短期間で文法を理解し運用する練習ができる。この際、短期間で文法を修得させるために、文法の説明は非常に形式的になる。

プログラムは、プログラミング言語の形式的な文法に従って書かれなければならない。ちょっとでも間違うと、コンピュータはプログラムを実行してくれない。従って、プログラミングの演習を行えば、形式的な文法を理解し運用する練習をすることができる。

+ 文書力の向上

第二外国語に限らず、語学を学ぶと母国語の文章力も向上するという。文章を読む力、文章を書く力は、言語に共通のものだからであろう。従って、プログラムを書いたり読んだりすることによっても、文章力が向上することが期待される。

+ 暗記力の鍛錬

いうまでもなく、語学には暗記の要素が多い。従って、語学を習えば暗記力を鍛錬することができる。

単語や熟語を覚えて語彙を増やすには暗記力が不可欠である。単語や熟語は、プログラミング言語でいうと、キーワードや基本演算子に相当するだろう。また、ライブラリの中の関数名や手続き名も、プログラミング言語の語彙を構成していると考えることができる。

+ 忍耐力の修行

第二外国語教育を忍耐力の修行といってしまっては身も蓋もないかもしれないが、一行ごとに必ず新しい単語が出て来るようなテキストを毎週何ページも読み進み、次から次へと出て来る不規則動詞の活用を暗記し、普通の名詞が男か女かを常に気にしていなければならないことは、忍耐力の修行以外の何ものでもない。

プログラミングも、忍耐力を養う絶好のチャンスということができる。特に、形式文法という非常に厳格な枠の中でプログラムを書かなければならないところがよい。

+ どのようなプログラミング言語

プログラミングの演習が暗記力の鍛錬や忍耐力の修行であるとするならば、プログラミング言語は何でもよい。むしろ、Common LispやADAのような難解な言語の方がよいとさえいえる。しかし、一生で一度のプログラミングとの出合いの場であると考えると、余りメチャクチャな言語ではなく、より教育的な言語の方がよいであろう。

例えば、MLやSchemaなどの関数型の言語は、教育的には非常によい言語だと私は思う。しかも、文房具程度には使うことができる。簡単な実験道具としても十分である。

* 計算機導入演習

かつて計算機が高価な貴重品だった頃、計算機を購入して運用するのは、大型計算機センターなどの専門の部署の選ばれた人々に限られていた。しかし、計算機が文房具としても使われるようになってしまったいま、専門家に計算機の購入と運用をやってもらえるはずもない。do it yourself、自分のことは自分でやらなくてはならない。しかし、パソコンやワープロくらいならば家庭電化製品と似たようなものであるが、ワークステーションともなるとそうもいかない。つまり、ワークステーションの技術は、残念ながら、馬鹿チョンのレベルにまでは達していないのである。

しかし、たとえ文房具として使うのにも、ネットワークのことを思うと、やはり、いまのところは、悪名高きUNIXワークスーテションい外には考えられない。実験器具として使うならばなおさらである。ところが、現在の非馬鹿チョンのワークステーション技術では、大学の研究室などでワークステーションを中心にして計算機とネットワークを運用していくことはとてつもない大仕事なので

ある。その結果、けがれを知らない大学院生や助手が計算機とネットワークのお守り役としてその貴重な青春を捧げるという現代の悲劇が至るところで見受けられる。

従って、「計算機導入演習」が教養課程の科目として存在すれば、彼らの苦労は少しは軽減されることが期待される。少なくとも、情報不足で右往左往しなくてすむようになるし、種々の実習まで受ければ、実際に計算機を導入する際にかなり役立つのではないだろうか。

以下では、「計算機導入演習」について、

+ 購入
+ 設備
+ インストールと保守

の三点を説明しよう。

+ 購入

いくら安くなったといっても、ワークステーションは1台100万から200万はする。これは、貧しい大学の研究室の年間予算くらいに相当する金額である。従って、どのような計算機をどこからどのようにして購入するかは、非常に重要な問題である。

素人が計算機を選ぶコツは、やはり、最も標準的で多くの人が使っている計算機を選ぶということだろう。例えば、ワークステーションならば、いまのところは、SUN(SparcStation)である。多少高くついたとしても、後のインストールや保守の苦労の少ない方を選ぶのがよい。

次にどこから買うかということが問題になる。たとえ値引き率が少々悪くても、やはり、SEが技術的にも信頼できて、しかも、面倒見がよく素人のつまらない相談にも親身になってのってくれるところがよい。コンピュータの販売会社には、大手、中小、外資系とあるが、一般的に、大手は技術的にはきちんとしているが融通はきかない、中小は技術的には劣るが面倒見はよい、外資系はSEの当り外れや入れ替わりが大きい、といったところだろうか。

+ 設備

パソコンならば家庭電化製品のようなものであるから、100Vの電源さえあればよいが、ワークステーションともなるとそうはいかない。特に、ファイル・サーバなどの上位機種を購入してローカルなネットワークを組んだりするためには、ある程度しっかりした設備が必要である。特に、広域のネットワークに接続したり家から電話で使ったりするために無人で24時間連続運転するには、防犯や火災に供えて万全の設備を準備しておかなければならない。

計算機を導入するための設備にはいろいろなものがあるが、何はなくとも電源である。計算機の電源は家電のように100Vとは決っていない。アメリカ製のワークステーションには125Vのものが多いし、200Vといったものもある。従って、各計算機の容量も考慮して適切な電源工事をしてもらうことが重要である。

次に床である。計算機室というところは、端末室も含めて、二重床にするのが一般的である。二重床の下の床には、電源やネットワークのケーブル類を走らせる。どうしても二重床にできない場合は、ケーブル類をなるべく固めるようにし、その上に被いをつけるのがよい。

次に空調である。特に、ワークステーションを導入して無人で24時間連続運転するためには、信頼性のある空調を用意しなければならない。空調が故障して部屋の温度が上がったときに電源が自動的に落ちるような仕掛けもあるという。なお、空調が暴走して計算機室が水浸しになる事故はよくあるので、空調の本体は計算機室とは別の部屋にあった方がよい。

次にネットワークである。現在でも、RS232Cやモデムの知識は基本中の基本であろう。それに加えて、イーサネットの知識は不可欠である。イーサネット・ケーブルの加工、トランシーバの取り付けは自分でできた方がよい。

最後に防犯設備について一言。不特定多数の人の入る端末室は、どろぼうの天

国になる恐れがある。計算機を盗まれないまでも、心ない人がシステムにいたずらをすることはよくある。計算機室をどう管理すべきかも非常に重要な問題である。

最後の最後に一言。計算機室、端末室は、必ず禁煙にすること。煙は計算機の天敵である。

+ インストールと保守

これが最も大変な仕事である。

ワークステーションは、買ったらすぐに使えるというしろものではない。使い方にあわせて、各種のソフトウェアのインストールをしなくてはならない。もちろん、OSのインストールなどは買ったときにSEがやってくれるが、PDSともなると、自分でやるしかない。また、ネットワークのインストールは特に大変である。ローカルなネットワークの敷設はSEがやってくれるかもしれないが、広域のネットワークへの接続はSEの感知するところではない。

インストールが完了しても、計算機の保守(メンテ)は、計算機が使用されている限り続けなければならない終身刑のようなものである。例えば、新しいユーザの登録などの仕事が典型的であるが、本当に大変なのは、ディスクやネットワークの見張りである。

ユーザの多い計算機では、ディスクがパンクすることがしばしばある。特に、ネットワークに接続している場合は、電子ニュースなどのせいで、誰も使っていないのにいつのまにかディスクが溢れてしまっていることがよくある。残念ながら、現在のワークステーションは、ディスクがフルになるとにっちもさっちもいかなくなってしまうので、ディスクの空きスペースが十分にあるかどうか四六時中気を付けていなければならない。

ディスクに関しては、ファイルのバックアップの体制も重要な問題である。どのような頻度で、ファイル・システムのどの部分を、どのような方法でバックアップするか、きめ細かな注意が必要である。

次に、ネットワークに関してである。現在のネットワーク技術では、残念ながら、ネットワークがトラブルを起こすことがしばしばある。ネットワークがトラブルを起こした場合、何がいけないのか、例えば、ハードかソフトか、自分か相手か、ケーブルかトランシーバか、などなど、トラブルの原因を同定するのが非常に難しい。また、ネットワークのトラブルは、いつ何時起こるか全くわからないので、ネットワークの見張りは欠かすことができない。

最後に、計算機、特に、ハードウェアが壊れてしまった場合の処置についても、ある程度教えておくべきであろう。家庭電化製品と同様に、大抵は買ってから一年間の無料保証が付くが、それを過ぎてからは、正式な保守契約に入っていないと、実費相当の修理代が必要になる。そのとき、たまたま予算が余っていたりすればよいが、なければ研究室全体がパニックに陥るだろう。正式な保守契約も結構お金がかかる(一年間で購入金額の数パーセント)ので、どちらにするかは、計算機の信頼度なども考慮して、適切な判断を下さなければならない。

* 計算機概説

自動車の免許を取るにも、交通法規だけではなく、自動車の仕組み、つまり、自動車のメカをある程度知っていなければならない。同じように、文房具であろうと、実験器具であろうと、計算機がどのような仕組みで動くのかを知っていても損にはならないだろう。

計算機のハードウェアからソフトウェアに至るまでをうわべだけさらっと大雑把に解説するのが、いわゆる「計算機概説」というような名前の付いた大きな階段教室にお似合いの眠たくなりそうな岩波新書のような講義である。表1に、「計算機概説」の項目だけを挙げておこう。

表1 計算機概説

ハードウェア

素子技術 --- トランジスタ・LSI

ディジタル回路 --- 論理回路・順序回路
入出力装置 --- キーボード・CRT・ディスク・SCSI

アーキテクチャ

CPU --- CISC・RISC
メモリ --- SRAM・DRAM
バス --- DMA転送・VMEバス

オペレーティング・システム

オペレーティング・システムの機能
--- タスク管理・データ管理・入出力管理
オペレーティング・システムの例 --- MD-DOS・UNIX
ネットワーク --- イーサネット・ISO

プログラミング言語

プログラミング言語の例 --- Fortran・Pascal・Lisp
形式文法 --- BNF・LL・LR
処理系 --- コンパイラ・インタープリタ・コンパイラコンパイラ
プログラミング・パラダイム
--- 関数型・論理型・オブジェクト指向

ユーザ・インターフェース

コマンド言語 --- shell
日本語入力 --- 仮名漢字変換
ウィンドウ・システム --- Macintosh・X

* 計算基礎論入門

数学の一分野で、集合や論理を扱う学問を、数学基礎論という。数学基礎論はいつも数学の中で異端視されている。それは、数学基礎論が、数学を研究するのではなく、数学という学問自身を研究の対象とするメタ数学だからである。文学史の研究が文学そのものにとって何の役にもたたないように、数学基礎論も数学そのものにとって何の役にも立たない。

プログラムの理論や計算の複雑さなどの計算に関する理論的な研究も、数学基礎論と似たような立場にある。しかも数学基礎論との関連が深い。従って、これらの研究は計算機を使って実際の計算をしようとする人にとって、何の役にも立たない。まさに、真の教養である。そこで、数学基礎論をもじって計算基礎論という名前にして、その入門を教養でやるのがよいだろう。表2に、計算基礎論入門の項目だけをまとめておく。

考えてみると、オートマトンやλ計算の概念は、集合の概念が数学にとって重要であるように、計算機を使う人にとっても教養として非常に重要であると思われる。これほど計算機が大衆化してしまったいま、大学の教養課程といわず、むしろ、高校で教えた方がよいのかもしれない。実際、微積分を習う日本の高校生にとって、λ計算などちょろいものであろう。

表2 計算基礎論入門

言語理論 --- オートマトン・文脈自由文法
計算可能性 --- 帰納関数・チューリング機械
計算の複雑さ --- NP完全・限定算術
計算の論理 --- λ計算・型理論・カテゴリ理論
プログラムの理論 --- 意味論・検証・合成
自動証明 --- 導出原理・帰納法