L I M E　'8 9

－学祭号－

K I T　コンピュータ部

目次　　ページ

マイクロマウス入門編　2
Making of
TWIN TENTRIS　8
言語作りにはしろう
（ま）　12
エジプト式分数　16
レイトレーシング　20
編集後記　38

written by Hanako

マイクロマウス入門編

ANKO

さて、今年は動くかどうか分からないが　マウスの概略と動かし方について説明します。

1）概略　－マイクロマウスとは－

マイクロマウスとは、迷路をセンサで探りながらゴールを目指すものである　そして、そのセンサで得た迷路についての情報を処理して　さらに早くゴールを目指す自律型のロボットである。

2）コンピュータ部のマウスの仕様

ＣＰＵ　：　６４１８０（Z80 C-MOS版として使用）
センサ　：　赤外線フォトセンサ
駆動力　：　ＤＣモータ　（ＲＥ－１４０）
ソフトウェア：Ｔｉｎｙ　Ｂａｓｉｃ

3）マウスの起動

まず、ＰＣ９８０１かＰＣ８８０１を用意してください。専用のＲＳ２３２Ｃケーブルをマウスのボードとパソコンとの間につなげます。マウスに電源をつけます。ＰＣの電源をいれ、ＢＡＳＩＣを立ち上げる。ＦＩＬＥ数を４くらいにとる。（２でもできます。）

ＢＡＳＩＣ上で、「 TERM "N81X" 」といれてリターンキーをおす。その後、マウスのリセットボタンをおす。

すると、Ｔｉｎｙ　ＢＡＳＩＣ　が立ち上がります。これで、マウスのＣＰＵにプログラムを入力できます。しかし、このマウスのＢＡＳＩＣはＴｉｎｙですので、使いにくい。なるべく特殊なコマンドは使わないように。

さて　マウスのセンサや、モータを動かすにはやるべきことが幾つかあります。まず、ＰＩＡをイニシャライズします。

```
10 OUT 65,00
20 OUT 64,00
30 OUT 65,04
40 OUT 67,00
50 OUT 66,255
60 OUT 67,04
```

これで、ＰＩＡが使えます。この後、６４番地を読むと、センサのデータが読み込まれ、６６番地に書き込むと、モーターが動きます。

読み込み例

```
70 A=INP(64)
80 PRINT A
```

書き込み例

```
90 OUT 66,A
```

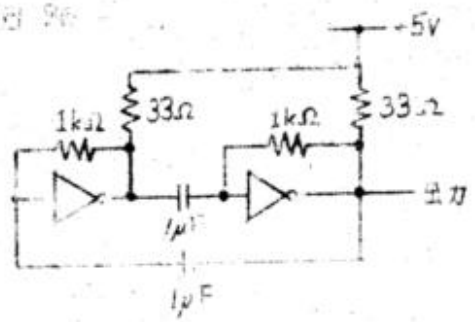
上のプログラムは、６４番地を読み、そのデータを表示し、６６番地に書き込むようになっています。６４番地のデータは、センサの反応のあったｂｉｔだけＬになっていて他はＨです。ハードに強い方なら分かるでしょう。まぁ、始めての方にはセンサの読み込みよりも、モータの動かし方の方がいいでしょう。

６６番地のｂｉｔは、０　左正転
１　左反転
２　右正転
３　右反転
４　未使用
５　未使用
６　未使用
７　未使用

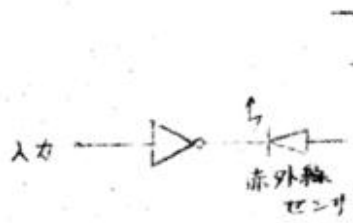
となっています。このそれぞれのｂｉｔをセットした時の１０進数の値を６６番地に書き込めばモーターが動きます。一度お試しください。

４）あとがき

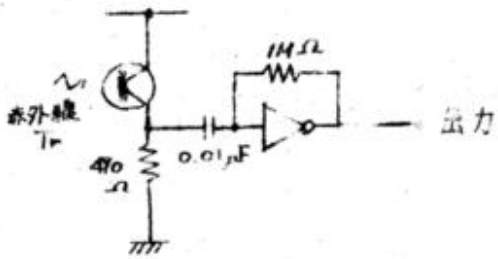
マウスのハード・ソフト共に今年は担当・製作しましたが、なんといってもどちらも、まともな資料がほとんどなかったのがつらかった。参考にしたのは、去年のＬＩＭＥに掲載されたＣＰＵボードの回路図（ＰＩＡのポートはここから解読した）と、センサの回路図だけである。特にソフトウェアの面では、試行錯誤の上作成したようなものである。その意味もあって、起動方法とＰＩＡについてだけでもこのＬＩＭＥに掲載しておこうと思った。全ては、将来の製作者のためである。



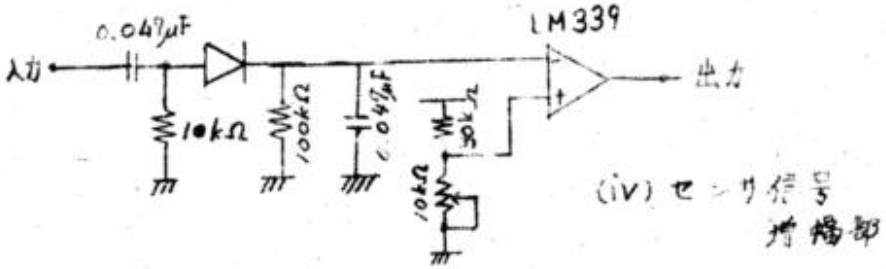
(i) 発振部



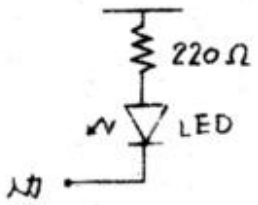
(ii) センサ
発光部



(iii) センサ
受光部



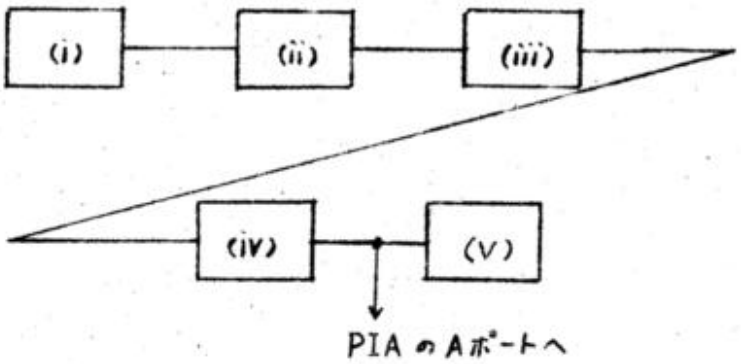
(iv) センサ信号
増幅部



(v) センサ出力確認部

◎ センサ回路部（続き）

接続図



回路 (i) (iv) は、センサ基板 I に

回路 (ii) (iii) (v) は センサ基板 II, III に 実装

基板図

MAKING OF 幻の TWIN TENTRIS

by ニュース・ステーション制作班

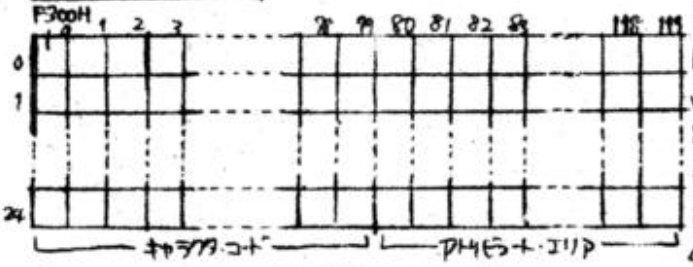
はじめに

京都工芸繊維大学合格決定後、何を作ろうかということで、「Nモード(N-BASICモード)で名作ゲームを再現してみよう」ということになり

・TETRIS　・上海　・アルカノイド　・ロードランナー

等の候補から、今年のゴールデンウィーク12番エ(工事やないっちゅーの!)しました。実はそれまでにBASIC+マシン語版を作っており、それの移植を流れがあったので、ゲーム本体は1日間ほどで出来上がってしまいました。そこで、ボード表示関係のアルゴリズムを説明しようと思いますが、その前に、Nモードのカラー表示にふれておこうと思います。

アトリビュートエリア



PC-8000シリーズのテキストV-RAMは左図の様に、120×25=3000バイトで構成していますので、色の付けかが特殊です。(一応80字モードで説明しますが40字モードは後述します)

120バイト中の40バイトで色を指定しますが、2バイト1組で

<カラム> <アトリビュート・コード>

の順に指定していきます。<カラム>=X、<アトリビュート・コード>=Cとおくと、

X桁目からコードCで色をつける　($0 \leq X \leq 80$)

ということになるわけです。たとえば、F350Hから

00 E8 02 48 0F C8

とセットすると、

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	……
白	―						赤								黄		

という風に色がつくわけです。よくI/O等昔の雑誌を見てますと、カラーをふんだんに使ってマシン語でゲームを作ると速くなるなんて書いてあるのは、上の例では、「7桁目から水色にしよう」とすると

00 E8 02 48 07 A8 0F C8

という風に挿入する形になります。これを汎用ルーチンでやろうとすると、

・挿入位置のポインタ
・挿入位置から後ろのデータをズラす　(ア)

C	色
08H	黒
28H	青
48H	赤
68H	紫
88H	緑
A8H	水色
C8H	黄
E8H	白

(ⅴ) といったことが必要になり、汎用に作ってあるROM内サブルーチンでは速度が落ちるわけです。又、書換回数が増えると色が変わらないということも起こり得るので、Nモードのカラーゲームでは、そのバランスをとるのに苦労されたのが先人達、ということになります。ところがテトリスはご存じの様にカラフルなゲーム。そこで

カラムを固定してしまえばいい!

という発想にたって、画面の初期化でアトリビュートエリアを

```
00 00 02 E8 10 E8 1C E8 1E E8 20 08 22 08 24 08 26 08 28 08
```

- 02 E8：スコアボード (SCORE:青)
- 10 E8：MISSの数を表示する部分
- 1C E8：スコアボードの右側
- 1E E8：ゲームボードの左側
- 20 08 〜 28 08：ゲームボードのブロック1個分にそれぞれ2バイトあてている

```
2A 08 2C 08 2E 08 30 08 32 08 34 E8
```

- 2A 08 〜 32 08：ゲームボード部
- 34 E8：ゲームボードの右側

という風に設定し、ゲーム中は

アトリビュートコードのみを書き換える

ためにすることで、画面の色処理をはぶいています。又、ゲームボードは全部■のキャラクタで埋めておいて、ブロックの表示は、このアトリビュートコードの書き換えだけで表示しています。

さて通信ですが・・・

RS-232Cポートを使い、専用データを双方でやりとりすると決めたまではよいのですが、(データのフォーマットは2転3転しましたが) TE-N-TRISに組み込むとこれが動かないっ!

Nモードで RS-232C回線を使うには、それを制御している

8251

と呼ばれるLSIを制御しなければなりません。詳しくは書けませんので、準備の手順だけ書きますと、(当然アセンブラでの話ですが)

```
CALL   0D14H        ← 8251のリセット
LD     A, xx        ← 細かいパラメータのセット
                      (ストップビット・パリティ・ボーレート等)
CALL   0ECDH        ← エラーフラッグ(8251内)のリセット
LD     A,(0EA66H)  ┐ 入出力インターフェースがカセットと共用
OR     20H         │ なので、RS-232C用に切り換えます
OUT    (30H),A     ┘
```

で出来ます。このルーチンを使って、任意の8bitのデータをやりとりするプログラムは正常に動いたので、TE-N-TRISへ組み込みます。次に実際のデータ入出力ですが、

20H : データの入出力

21H : コマンド出力/ステータス入力

の2つのポートを使って、以下の様に書きます (次頁へ)

「入力」
```
LOOP:  IN A,(21H)   ┐ bit1=1になるまで
       AND 2        │ 待つ
       JR Z,LOOP    ┘
       IN A,(20H)   ← データ入力
       RET          (ストップビット以外のエラーのチェックは略)
```

「出力」
```
       PUSH AF
LOOP:  IN A,(21H)   ┐ bit0=1になるまで待つ
       RRCA         │
       JR NC,LOOP   ┘
       POP AF
       OUT (20H),A ← データ出力
       RET
```

となります。(基本的な考え方で、実際は若干異なります) 前述のテストプログラムはこれで動いたのですが、これが TE-N-TRISに組み込むと動かない!
原因としては

- TE-N-TRISの入/出力サブルーチンを呼ぶまでにタイミングがかかりすぎてデータが落ちている
- クロスケーブル(自作)がノイズをひろってる
- 使っているICの片側が「ターミナルモード」のテストの為に時々変な動作をしていたのでひょっとしてICのI/Fの不良?

等があげられますが、3番目はほとんど苦しまぎれ、2番目はテストプログラムが動いているところを見るとおそらく違うだろう、ということで 10/20 日より5階現在このテストは動いていません。困ったことに… なにせ、RS-232Cでケーブルに信号が出ているかどうか調べる手段が不明なものですから… 情けないことですが。
というわけで、現在は原因検討中であります。

以上 報告 終わり

言語作りにはしろう！

BASIC愛好会 会員No.6
吉賀 正博

「春までに、Tiny BASICを作っとけ。」と部長のこんちゃんに言われたのは、昨年の冬頃だったろうか。その頃私は、Tinyではあるが For 文を作り終えて気分が良かったところだった。当然、快く「うん、わかった！」と、あっさり答えてしまった。そして泥沼にはまりこんでいった……。

まぁ、冗談はさておき、いまから言語作りの方法を簡単に書いてみよう。言語作りは難しいのでは、と思われるでしょうが、難しくしようと思えば難しくなるし、簡単なものでいいのなら簡単にできます。（私の場合、C言語を覚えて、最初に作ったプログラムが8 Queenで、2番目がFor文だったんですから。え？説得力に欠ける？すいません。）

さて、本題に入ります。コンパイラを作るにせよインタプリタを作るにせよ、言語作りの基本は同じで、字句解析→構文解析→意味解析→コード生成（インタプリタなら実行部）と、この4つのブロックを作ればできあがりです。各ブロックの仕事は大体わかるでしょう。簡単に述べると、字句解析で、1つずつ字を読んできて字句にします。（ifなら'i'と'f'とはなく'if'として認識する。）できた字句を解析し、意味を理解し、コードを生成する。（インタプリタなら実行する。）というところです。各ブロックは、本当はもっと細かいことをしているのですがとりあえず省略します。

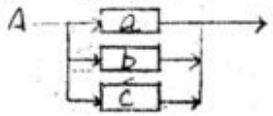
こんな言語が作りたいというのが決まれば、まず設計図を書きましょう。これは、EBNF記法とか、

構文図などが使われます。EBNFは、ALGOL60の定義に使用されたBNF（Backus-Naur-Form）を改良した拡張BNFのことであり、例えば、

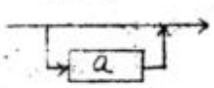
S = AB
A = x | y
B = z | w

のように使われ、SはABで定義されるとまもなく、|はまたはという意味で、結局、上の定義はよりSは xz, xw, yz, yw の4種類の文が生成可能である。|の他にもいろいろあって、{}は0回以上のくり返し、[]はあってもなくてもよいことを意味しています。EBNF記法は、直観的にわかりにくいので、かわりに構文図がよく使われます。EBNFと構文図の対比を以下に示します。

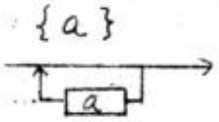
A = a | b | c



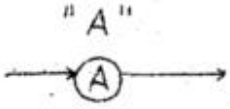
[a]



{a}



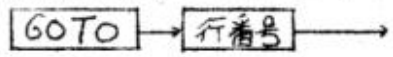
"A"



このように、構文図の方がわかりやすく、簡単に覚えられると思いますので、どちらかといえば構文図をおすすめします。

これで、基本は終わりです。あとはやってみればなんとかできるものです。ちょっと難しくてできないようなところがあれば「それは仕様です」といいきってしまえばいいのです。（ちょっと無責任だけど）それでは、実際にBASICを（当然Tiny版だけど）作ってみましょう。

まず、仕様書をつくりましょう。変数名は1文字だけ数値は10進数のみとすれば簡単でしょう。それから、中間言語の生成もなしにした方が楽ですね。AUTO文もUNLIST文もDEF文もなし。LIST文は、行番号指定なしでやった方が最初は楽でしょう。マルチステートメントもなしでいいでしょう。だいたい、仕様が決まれば、構文図を書きましょう。GOTO文なら、



という具合に、以下全部書きましょう。ここでミスると構文解析ブロックを作るときに大変になります。

さあ、いよいよプログラミングです。BASICのメインルーチンは実に簡単で、1行入力され、その1行が数字で始まってるとプログラムが入力されたとみなし、その1行を格納する。それ以外のときは、ダイレクト命令なので、その1行を実行する。以下、そのくりかえしだけです。よって、C言語でメインルーチンだけを簡単に書くと

```
main(){
    while(1){
        1行入力;
        if(先頭が数字)
            プログラム格納;
        else
            ダイレクト命令を実行;
    }
}
```

となります。他にもいろいろしなくてはいけないのですが、最初はこれで十分です。

次に、プログラム格納ルーチンを作るのですが、ここは1つ、ポインタを用いたリスト構造でプログラムを保存しましょう。リストの連結・削除など、大変勉強になりますし、配列に比べるとやっぱり便利です。

あとは、ダイレクト命令実行部だが、これは実に簡単。1語読んできて（字句解析）、それが命令であればそのサブルーチンへとぶ。違えば Syntax Error だ。各サブルーチンでは、先程書いた構文図をもとにしてプログラムすればいいだけだから楽である。これで、大体は終ったような気になる。

が、一番大変なのが、"式評価ルーチン"である。これは式が書かれた文字列を評価して答を返すサブルーチンである。これができたらもう、プログラム全体の6～7割はできたといっても過言ではないだろう。ここで式評価の1つめの方法は逆ポーランド記法に直すやり方がある。コンパイラを作る場合はよくこの方法が使われるが、インタプリタの場合、一度逆ポーランド記法にするためにテキストをよみ、さらにそれを見て計算をするというのは二度手間となり、ちょっとナンセンスだ。2つめの方法は式をEBNF記法で定義し、1つ1つを関数とするやり方。3つめは、スタックを利用し、優先順位をうまく利用する方法である。詳しくは、文献をみるか（1,2冊ほど前のクラづの日誌（いわゆる日記帳）をみて下さい。私の場合は、3つ目の方法を使いました。

以上でプログラム完成です。あとは、命令を拡張するなり、追加するなりして下さい。いずれ自分がBASICにぱっぽつはまってゆくのに気付くでしょう。めざせ明日の構造化BASIC！

P.S. この文を読んでるとボトムアップ方式でプログラムを作ったと読みとれますが実は、おもいっきりボトムアップ方式で作ってしまいました。申し訳ありません。

突然ですが

エジプト式分数

よしが

今年の松ケ崎で音楽プログラムを担当しました。4分音符、8分音符、16分音符などに加え、3連符を表現するために、6分音符、12分音符などがでてくると、もう、頭がパニック寸前になるまで、分数の計算の連続です。そう、コンピュータ・ミュージックは、分数計算、しかも、分子が1の分数計算をよくやるんですよ。そこで、思いだすのが、このエジプト式分数です。（ちょっと強引かな？）

エジプト式分数とはいったい何でしょう。古代エジプト人は、当然1より大きい分数について十分理解はしていたものの、1つの数としては表現しなかった。つまり、単位分数、すなわち、正の整数の逆数のみを扱っていたのである。たとえば、5/6の場合、そうは書かずに、1/2 + 1/3と書いたのである。だいたいエジプト式分数についておわかりいただけたであろう。以下、このエジプト式分数について、いろいろ述べていきたい。（いっことわっておくが、ここでは1を超えない分数、真分数のみ考えていくことにする。）

ある真分数 a/b を、エジプト式分数に展開する方法の1つにフィボナッチの方法がある。この方法は、実に簡単で、ある真分数 a/b を展開するとき、展開の初項は a/b をこえない最大の単位分数で、第2項は、その残りをこえない最大の単位分数である。以下、この操作をくりかえすと、残りの分数は確実に減少し、有限回で必ず終わることが証明されている。この方法は、「貪欲算法」と呼ばれている。それは、各回の過程で可能な最大の分数を選んでいくからである。

では、いまの強欲算法を用いて、2/3を展開してみましょう。2/3をこえない最大の単位分数は1/2だから、2/3から1/2をひくと、1/6となる。つまり、2/3=1/2+1/6で、うまく展開できたようです。みなさんもいろいろな数でためしてみて下さい。

ところが、この強欲算法には恐しい欠点があるのです。それは、いつでも最良の展開式を作りだしえないということです。最良という明確な定義は無いが、項の数が最小である、もしくは、分数中の最大分母を最小にする、というこの2つが、一般的に最良とされている。(例えば、3/7の展開では、1/4+1/7+1/28が最小の項数という意味で、1/6+1/7+1/14+1/21が最大分母を最小にするという意味で最良である。)しかし、この両方の意味での最良という形にならない場合が、この強欲算法を用いると生ずるのである。例えば、この算法を5/121に適用すると、5/121=1/25+1/757+1/763,308+1/873,960,180,913+1/7,638,092,437,828,241,151,744という級数が得られるが、これは、パッと見ただけで、最悪といえそうだ。5/121は1/33+1/121+1/363とも展開できる。前者と後者を比較すると、後者の方が最良といえよう。

とこ次に、a/bの形の分数をどのようにすれば、うまく展開できるだろうか、という問題について考えていきたい。

いきなり、a/bではうまくいかないので、とりあえず、1/bについて考えることにする。これは、強欲算法を用いるとうまくいく。つまり、1/b=1/(b+1)+1/{b(b+1)}となる。また、

このことから、エジプト式分数の展開の可能性が無限にあることがわかる。例えば、1/2は1/2 = 1/3 + 1/6となり 1/3も同じように展開できるので、1/2 = 1/4 + 1/12 + 1/6となり、単位分数の級数に表わす表現はいくらでも続けられる。

次に2/bについて、bを2n, 2n+1にわけて考えると 2/2n = 1/n, 2/(2n+1) = 1/(n+1) + 1/(2n+1)(n+1)と2項以下で展開できる。同様に3/bのときは、bを3n, 3n+1, 3n+2にわけて考えてやればよい。

- b = 3nのとき
$$\frac{3}{3n} = \frac{1}{n} \quad (1項に展開)$$
- b = 3n+2のとき
$$\frac{3}{3n+2} = \frac{1}{n+1} + \frac{1}{(3n+2)(n+1)} \quad (2項)$$
- b = 3n+1のとき
$$\frac{3}{3n+1} = \frac{1}{n+1} + \frac{2}{(3n+1)(n+1)} \quad (3項以下)$$

(さらにnを2k, 2k+1にわけると
n = 2k+1のとき
$$\frac{3}{3n+1} = \frac{3}{6k+4} = \frac{1}{2(k+1)} + \frac{1}{(6k+1)(k+1)}$$
となり、2項に展開できるが、n = 2kのときは3項となる。)

よって、3項以下で展開できる。a/bのとき a項以下で展開できることは容易に証明されるが、3項以下で展開できるだろうか？残念ながらこれはまだ、証明されてはいない。4/nのとき、nの値が非常に大きな値まで確かめられてはいるが、4/nであっても証明はされてはいないのである。私自身も、ここ数日ではあるが、証明しようと試みた。上記と同じようにb = 4n, 4n+1, 4n+2, 4n+3とすると うまくいかない ときうまくいかず それを

さらに $b=12k-7, 12k-7, 12k+1$ に分けると、$c=12k+1$ のときうまくいかず、それをまた $k=2l, 2l+1$ の場合で考えると、どうしても、$b=24l+1$ のとき うまく展開できなかった。(まあ、頭のバカ数学者の先生たちにできないことを、私ごとき人間ができるわけないのだが……でも悲しい。)

さて、最後に問題を1つ。ある男が所有している11頭の馬と、長男に1/2、次男に1/4、末の子に1/6ずつわけるにはどうしたらよいでしょう。という問題があるでしょう。(これが、問題じゃないよ。)これは、1頭の馬を加えて12頭にし、そこではじめて1/2、1/4、1/6ずつわけてやると、それぞれ6、3、2頭ずつ馬をもらい、後から加えた1頭は残って、めでたしめでたし、という話でした。(いまさら、いちいちいわなくても誰でも知ってるよね。)他にも、こんなおもしろい分け方の変種が、いろいろ無限にあるように思われるが、じつは数通りしかない。それは、ディオファンタス方程式 $n!/(n+1)=1/a+1/b+1/c$ を解けばでてくる。ここ、ここでようやく問題です。それは何通りあって、どんな値でしょう。(え!この問題のどこがエジプト式分数に関係あるか?!方程式の右辺がちゃんとエジプト式分数、単位分数の和になってるでしょう。納得しましたか?)理論的に考えるとできると思うよ。コンピューターを使って、計算させてやるのも1つの手だね。分数を使ってるのだから分母と分子に分けてやると誤差もでないから、簡単にプログラムできるよね。(だんだん文章が変になってきたな~。)さて、問題をだしたまじゃなんだから、解答を示そう。と思ったが、書くスペースがない!! それでは、おさらばヨ~ナラ~!(←無責任。)

レイトレーシング

生産機械工学科 3回生
こんちゃん

テレビのニュースのオープニング画面なんかは、近ごろほとんどCG（コンピュータ・グラフィックス）ばっかしみたいになってますね。あんな絵を自分でプログラム組んで描かしてみたい！　それだけのことで始めたものの、そのスジの本やら雑誌をあさってみても、行列やら何やら、よーわからんことばっかり載ってて、結局座折してしまった。なーんてなことはありがちですねの私もそーでした。すいません。　そうかといって売ってるCGソフトを買ってきて、それを走らしてみても、何の満足も感じない体になってしまった私としましては、ここはいっぱつ自分で1から理くっコネてみて、自分の理くってCGのプログラムをでっちあげてしまおう。などと大胆な行動に出てしまったのでした。そーゆーわけで、私のでっちあげたプログラムは、ほとんど独断と偏見のかたまりみたいなもんです。そのヒやってることも幼稚で高校の数学ーどの知識しか用いておりません。それでもいっちょ前にCGっぽい絵がかけてしまうから笑っちゃいます。　まあ前おきはこのへんで、CG講座のはじまり、はじまり！。

上(↑)の所で言いわすれてますが、一口にCGといってもいろいろあります。私が描いてみたかったのは、あの写真のよーなリアルな絵のやつです。あーんな絵はどないしたら描けんねやろ？　と思って調べてみると、「レイトレーシング」とゆー手法でかいてるんや　ゆーことがわかりました。これはもうレイトレーシング　略してレイ・トレ　これしかないな思たわけです。

それでは極々簡単にレイトレーシングの原理をみてみましょう。レイ・トレとは？　読んで字のごとく、光の軌跡をたどっていくんですが、ゴチャゴチャ書いてみて

も解かりにくてーなんで、具体的に考えてみましょう。
とりあえず「球」がカンタンなんで それで考えてみます。

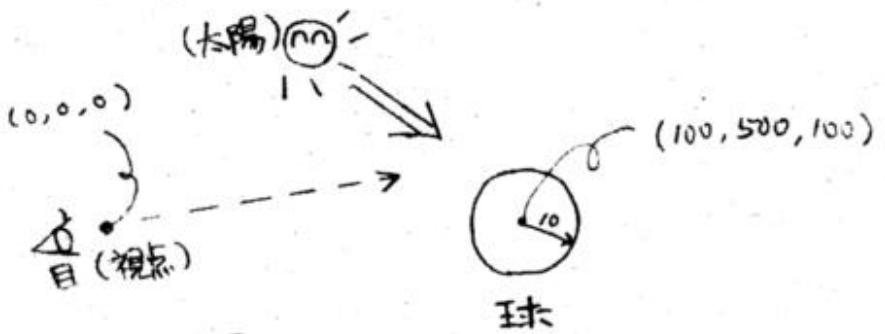


図1

まず3次元空間の位置は(X,Y,Z)の直交座標で考えることにします。 ほんで図1のような設定で球を見たとき、どんな風に見えるのか?ということを考えます。カンタンのため、視点を原点(0,0,0)に置いて、視線の方向(顔(目)の向いてる方向) 図の→はY軸方向とします。球の中心の位置は、仮に(100,500,100) 半径は10とします。

画面に球を表示させたいわけですが、パソコンのグラフィック画面の座標なんかはたいて一図2みたいな感じで2次元なわけです。(あたり前) 最大値はとりあえず255,211としときます。(MSX2の場合ほんで) ついでにカンタンのため(こんなことばっか言ってますが……) 1つのドットの縦横の比は1:1の正方形とします。 それでは、ここで次のページ図3のようにこの画面をだいたんにも球と視点のある3次元空間においてしまいます。(何をしるんですが!早くおきなさいなどと言ってもおきようがありませんので、各自頭の中で置いて下さい。)

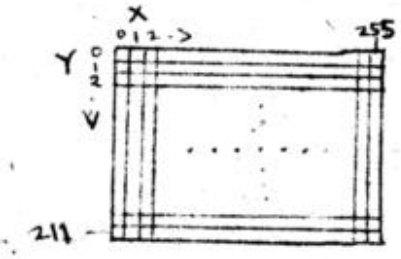


図2

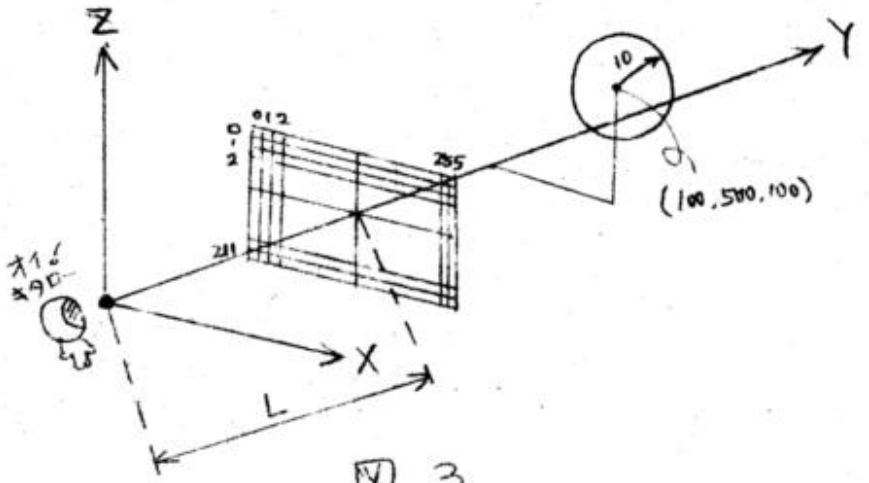


図 3

次に、このパソコンの画面上に球のシルエットを白く浮かびあげることを考えてみましょう。どうすれば、そんなことができるでしょうか。よく図3をながめてみますと、次のようなことがわかります。

1) キタローのお父さんの目（顔？）に入ってくる光は、空間の原点を通る直線と考えられるので、あらゆる方向から無数の光が入ってくる。
2) その光（直線）のうちY軸上（0, L, 0）に置いた、パソコン画面内を通っていくものを考えると、その光は画面と1つ窓を通して、お父さんの目に入っていく光と見なせる。
3) どーせ最終的にお父さんが見てる絵はパソコン画面に描くんだから、画面上の1つのドットに対して、そのドットの中央を通る光り1本だけでよい。つまり 255×212 コの直線だけを考えればよい。

ここまで来れば、あとはらく!!（何かごマカしてますか？）パソコン画面の[100,80]の位置のドットについて考えてみると、そのドットの位置は 3次元空間では どーなるでしょう？ 1ドットの長さを 3次元空間でも同じく1とすると、残は

画面　　空間

[128,106] → (0, L, 0)
[0, 0] → (-128, L, 106)
[255,211] → (127, L, -105)

みたいな感じなんで、[100,50]は

$$(\underline{100}-128,\ L,\ 106-\underline{50}) = (-28, L, 56)$$

となります。
そこで この画面上の点[100,50]の まん中を通って しかもお父さんの目に入る光は？ とゆーと (-28,L,56)と(0,0,0)を通る直線となるわけです。つまり

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} -28 \\ L \\ 56 \end{bmatrix} \cdot S \ , \ (S>0)$$

さて、やっとレイトレの本筋なんですが、その光をお父さんのいてる原点から出発して、光の進行方向と逆にたどっていくわけです。その直線がもし球と交わっていたとしたら、その光は球の表面から出たものとみなして、画面上の[100,50]のドットを白くぬります。もし交わってなかったとしたら、その方向には何んもないとみなして[100,50]のドットは黒くぬる。このそーさを画面上の全てのドットに対して ほどこしてやると、画面上に球のシルエットが(ベタぬりではありますが)浮かんでくるわけです。
それでは、球と直線が交わってるかどうかは、どうしたらわかるか？ これは高校の数学ですね。今球の表面は、

$$(X-100)^2+(Y-500)^2+(Z-100)^2 = 10^2$$

を満たしてますから、先ほどの直線の式と連立させて、X, Y, Zを消去すると Sの2次方程式となります。

(Lは定数です。) だからこの2次方程式の解があれば交わってるし、なければ交わっていない。これをコンピュータに解かせればよい。とゆーことです。

また直線の式の(-28, L, 56)のベクトルを先に単位化しといてやると、解があったときそのSは、視点から出発して、球表面までと、さらに球をつらぬいて出てきた裏面までの2つの距離を表してくれます。

それから何のイミもなく最後まで、ほったらかしにしてた「L」ですが この値をいろいろ変えてやると、絵のパースが変わります。つまり絵のひずみ具合が変わってきます。図3を見ればよくわかりますがLを小さくしてやると、広角レンズで覗いたような感じになります。

さてこれまでの話では単に画面上に白い丸が描けるだけです。それやったらサークル文で円描いてペイント文でぬりつぶしたんと同じやないかっ。そらそーですけどまあまあここからが本番です。

今までは、目から出発したある1つの光(直線)をたどっていって、球にぶつかれば白、何んもなかったら黒と、単にON-OFFでしたが、今度はその白にしてた部分に表面の濃淡をつけることを考えてみましょう。

そのためにまず、照明を設定しなければなりませんが、ここでは簡単のため太陽光線(平行光線)とします。その方向ベクトルを仮に(10, 100, 10) これを単位化しといて($10/101$, $100/101$, $10/101$)とします。

次に先ほどの2次方程式の解より、視点からぶつかった球表面までの距離がわかりますので、元の視点からの直線の式に代入して、ぶつかってる球表面の位置の座標がわかります。ではその点付近の光の状態を考えてみましょう。

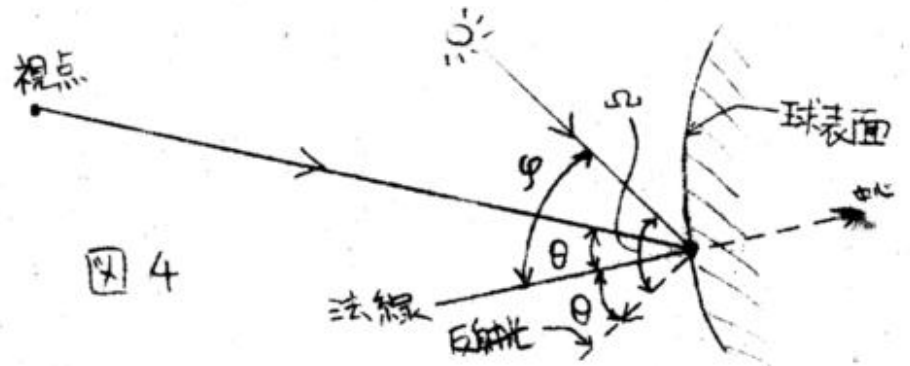


図4

図4をみてみましょう。球表面上の1点が決まれば、その点での法線ベクトルは、球の中心の座標を引き算して求まります。視点からの光の反射光のベクトルは、入射光と法線ベクトルに対して対称なので、これも求まります。

それでは話をもとどうりにして、太陽から光が出発して、その光がどうなって目に入るかをたどってみることにします。太陽から出発した光は、球表面上の1点にたどりつきます。そこで乱反射して、その光の1部が視点方向に進んで目に入ることになります。その乱反射して、目の方向に進む光の強さを図の角度 φ と Ω によって決定してやるのです。

私はややこいことは よーわからんので(←いばりきれてる。)

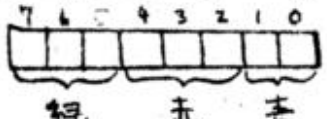
$$bright = C_1 + C_2 \cos\Omega + C_3 \cos\varphi$$

(C_1, C_2, C_3 : 定数)

とか なんとか式をでっちあげて、計算してみました。ははは このC1 C2 C3の定数をいじると、表面の感じがいろいろかわるでしょう。ははは (←えーかげんになってきた)

さて、これではまだ白黒の濃淡を計算してるにすぎません。やはりカラーで色をつけたいもんですが、それは簡単要するにBRGの3元色について、定数C1 C2 C3を変えて計算して、青の濃淡、赤の濃淡、緑の濃淡を出せばOKです。ここまで来て、実現段階で問題となるのは、そんなよーな カラーで濃淡のあるドットをパソコン画面上にどーやって出すか? 私は98とMSX2の2機種で試してみました。

まずMSX2は簡単です。スクリーンモード8というグラフィック画面は、分解能は256×212で1ドットにつき256色出ます。色の指定は0～255ですが、2進で

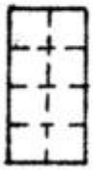


というビットの構成です。ですから、計算結果をB, R, Gという変数に0～7の8段階の値で用意すると、カラーのデータは

$$DT = G*2^5 + R*2^2 + INT(B/2)$$

です。

98では もともと 640×400 の分解能に8種類の色しか出せませんので.



このように8コのドットでひとまとまりのピクセルと見なして. 分解能は 320×100 となります。1つのピクセルに出せる色は. BRG 3元色の1つについては. 8コのドットのON.OFFで考えて. 9階調。3元色分のデータを演算することで. 9の3乗で729色出せます。

さてレイトレの概略はこのくらいにして、ここらでもっと現実的な話に移ります。私はレイトレーシングのプログラムをC言語で記述することにしました。そんな言語を選んで使い分けるほど多くの言語に通じてるわけではありませんが、BASICではあまりに遅い、その上データ型も弱すぎてやめといた方がよさそうです。パスカルでは? これは昔 turbo pascal を98上で動かしてレイトレのまねごとのよーな(今もまねごとではありますが)プログラムを書いたことがありましたが、まあこれはわりと良かったんですがタイプ量がやたら多くなったのと、turboの性格上メモリがけずられて、分割コンパイルすることを強いられました。ちんでデバック段階で非常に苦しんだ(turboにもかかわらず)のを思い出します。あと私の性格上、同じことをやりたくない。新しいことをやりたい。とゆーのもありました。Lisp では? 私はMu-LISPくらいしかLISPは知りませんので えらそーなことは言えませんが ビーもカジンのグラフィックが弱そーですし、なんといっても、ムツカシそう。(なさけない) アセンブラ?でもできそーですけど、とーってもしんどいことになりそうなんで、Cで書きました。(まともな理由が1つもないっ)

言い訳はこのくらいで、とにかくC言語(MS-C Ver4)で書くことになりました。それからもう少しアニメ作りのシステム全体について触れてみます。画面データはPC-9801 vm21で計算させるわけですが、画面の絵をビデオ信号に変換するのは非常に困難なものがあります。そうした関係上、計算したデータをシリアルインターフェイス(RS-232C)を通して

MSX2に送ってやることにし、MSX2からは直接ビデオ信号が出ているので、その画面をビデオデッキでコマ録りしてやることにしました。(図5参照)

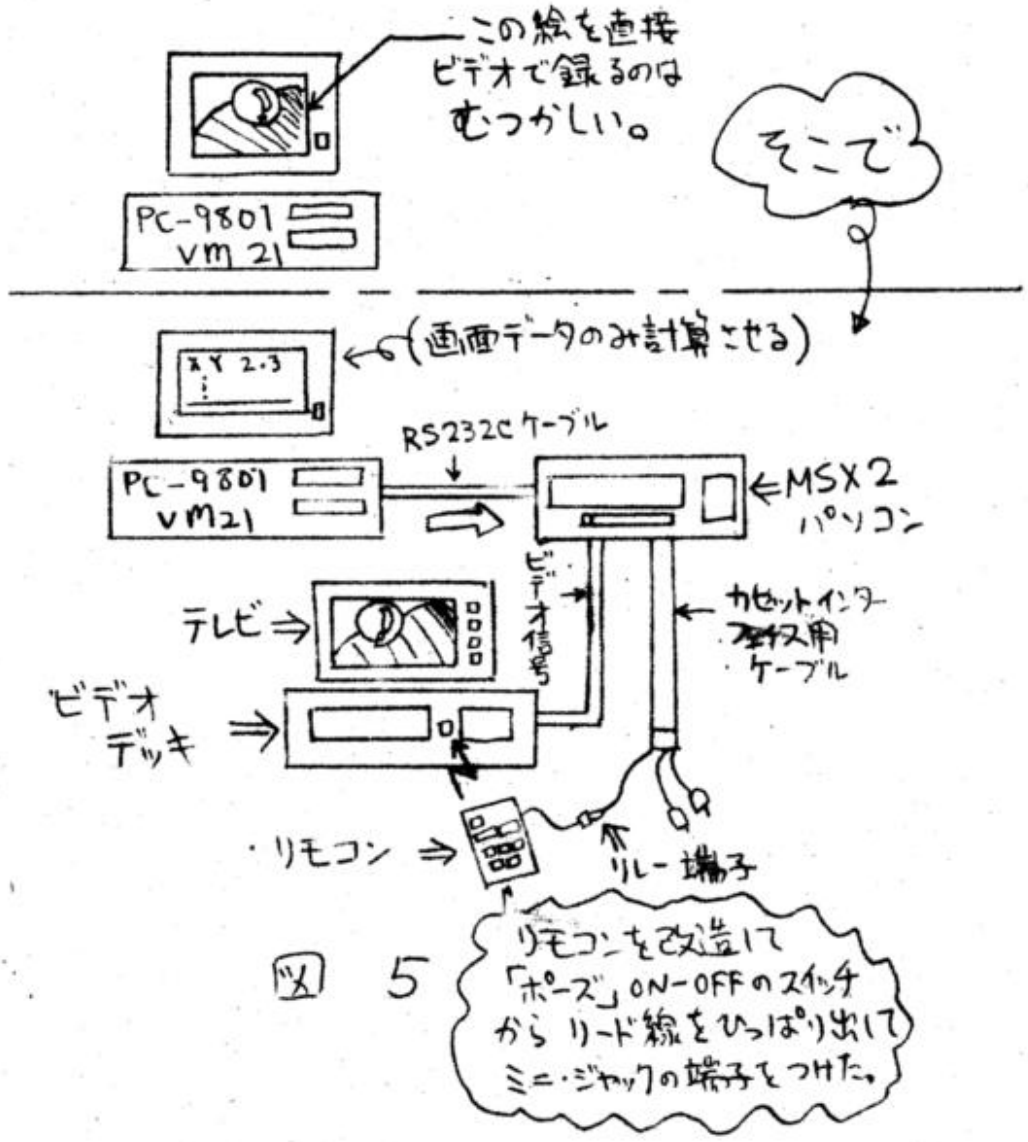


図 5

図5の下の様なシステムで アニメをコマ録りしてゆくわけですが この関係上、98の上で計算される画面データは MSX2の仕様に合わせてやれば良いということになります。それでは そのあたりのプログラムの実際にふれてみましょう。

MSX2画面は スクリーン8モードで使用するので 256×212ドットで 1ドットあたり256色の表示が可能です。しかし 画面のデータをチェックする段階で いちいちMSX2に送るのは手間がかかるため 98上でも表示できると便利です。そーしたことから 画面の分割は 256×200 ということにしました。

その分のバッファを MS-C内で

unsigned char vram[256*200];

として、確保してやって、そこに計算されたデータをためてやります。全データを計算し終わった所で そのデータを圧縮して ファイルに送り、1枚分の絵のデータの完成です。同様の操作をくり返し 全部のコマのファイルができた所で 順次MSX2へ ファイルのデータを解凍した後 送ってやり MSX2の内蔵のリレーを ON OFFして コマ録りしてゆきます。

98でやってることは 要するにこの vram[] に書きこむデータをひたすら計算してるわけです。先程説明したレイトレの大まかな原理では 触れていませんでしたが 実際のレイトレでは 球などの単純な図形ばかりを表示するわけではありませんので、複雑な図形も それな

りの工夫をしてやらなければなりません。そこで いわゆる「プリミティブを組み合わす」という概念を導入して実現させます。これは、「球」、「円柱」、「平面」、「円すい」などといった図形をプリミティブと呼び、それらの表す領域を 3次元的に AND, OR といった論理演算をほどこして、複雑な図形(立体)を表現する手方です。

例えば、図6の 球A, 球B で、球Aの内部と球Bの外部の領域の AND(論理積)を考えると

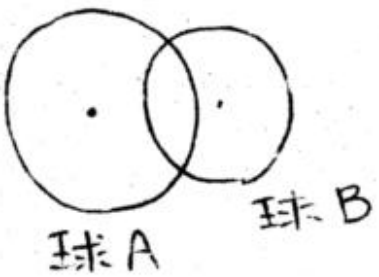


図 6

図 7

図7のよーに 球Aの表面が 球Bによって えぐられた図形となります。

この都合から、プログラムでは、プリミティブ 個々を表す。データと、その論理演算の手順を表すデータが必要になってきます。実際にプリミティブを どう表したかとゆーと、

```
struct pdattag {  /* プリミティブ データ タグ */
    char        id;     /*  id  */
    struct xyz  pnt;    /* 中心 */
    struct xyz  vek;    /* ベクトル */
    float       rad;    /* 半径 */
```

```
    float    col[3];  /* カラー */
    float    hlt    ;  /* ハイライト */

    char     knum   ;
    char     kflg   ;  } ☆
    float    ks[2]  ;
} pdat[PDATMAX];
```

このよーな構造体を宣言しました。

idは 円，円柱，平面，円すいの どれであるかを区別します。

中心とベクトルと半径は
円の場合は中心のみ有効，ベクトルは無視
平面の場合は 平面上の1点と法線ベクトル
といった具合に必要なもののみ有効とします。

なお，構造体の xyz型は

struct xyz {float x,y,z};

と宣言されており，3次元の座標とベクトル用に使います。

あとの カラー，ハイライトは，最終的に vram[] に書きこむ データを計算する段で参照し，その他☆は，プリミティブ同志の演算の段で利用します。

プリミティブ間の論理演算 手順を表すデータは，手ぬきで int unit[UNITMAX] と単に pdat[No.] の No.を ずらずらっと並べることにしました。

これらの図形のデータを元に、1つの直線(光)との交点をそれぞれのプリミティブに対しおのおの計算します。その結果は概念的には

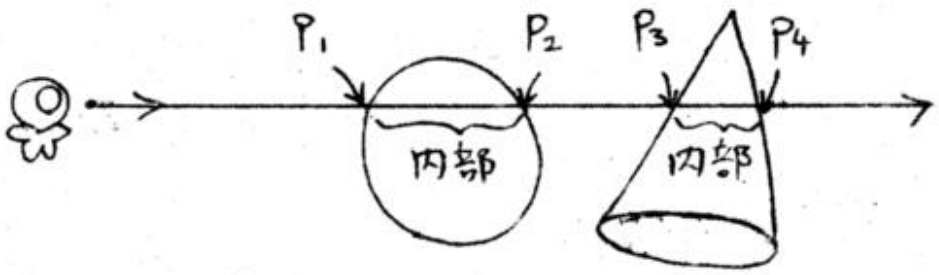


図8

図8のように、いくつかの立体を視線によってくし刺しにされたようなものです。1つの立体に対するこの結果のうち必要な情報は先程のプリミティブデータの中の所にいったん格納します。具体的には

pdat[n].knum には交点の数(平面は0or1, 球は0or1or2)

pdat[n].kflg には視点から出発した時点でその立体の内部にあれば1, 外部にあれば0とします。

pdat[n].ks[0or1] には、視点の方から出発して順番にぶつかる交点までの距離を表します。

この $\begin{bmatrix} \text{pdat[n].knum} \\ \text{〃 .kflg} \\ \text{〃 .ks} \end{bmatrix}$ の計算をした後、unit[]

の論理演算手順にしたがって、これらのデータのANDをとったりORをとったりして、最後的に図9のよーな

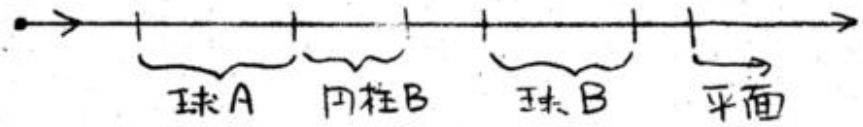


図 9

くし刺しのデータを得ます。(たとえば)
ここまでくれば、見えているのは、視線をたどって最初にぶつかる 球Aの表面であることがわかるので、法線ベクトル、その他光の方向などから その点の色の濃さ、3元色の割合を算出します。

実際にちまたで やってる レイトレーシングでは視線から出発して物体にぶつかると、そこから反射して さらにトレースをつづけることで、鏡面のよーな表面を計算したり、ぶつかった所から屈折して内部を通過してゆくことで、半透明な物体を表したりするわけですが、今回は計算時間の縮小のため、このよーなことは行いませんでした。その結果 いまいち絵が ベタっとした感じになってしまったり、光があたるはずのない所に影ができなかったりすることになりました。

ともかく。以上のよーな手順を踏みまして、vram[3]上へ データを作ってしまうわけです。

VRAM上にデータができると、次にそれをディスクに落とすわけですが、そのままでは 256×200=51.2[illegible]
になってしまって、1Mのディスクに約20枚しか絵が入らなくなってしまいます。そこで圧縮して保存するわけですが、今回は圧縮プログラムをてきとーにでっちあげて作ってみました。かんたんなやり方で、あまり圧縮率はよくなかったようですが、それでも「自分で作ってみる」のが好きな私なので満足してしまいました。だいたいの原理は図10のとーりです。

```
----A B C D D D D D E F G----
          ⇓あっしゅく
----A B C D D 3 E F G----
```

図10

つまり、圧縮されたデータで、同じ値が2回連続して表われると次にくる値は、それ以降その値が続く回数を表すものとします。

ではそのフローチャートを次の図11,図12に示します。

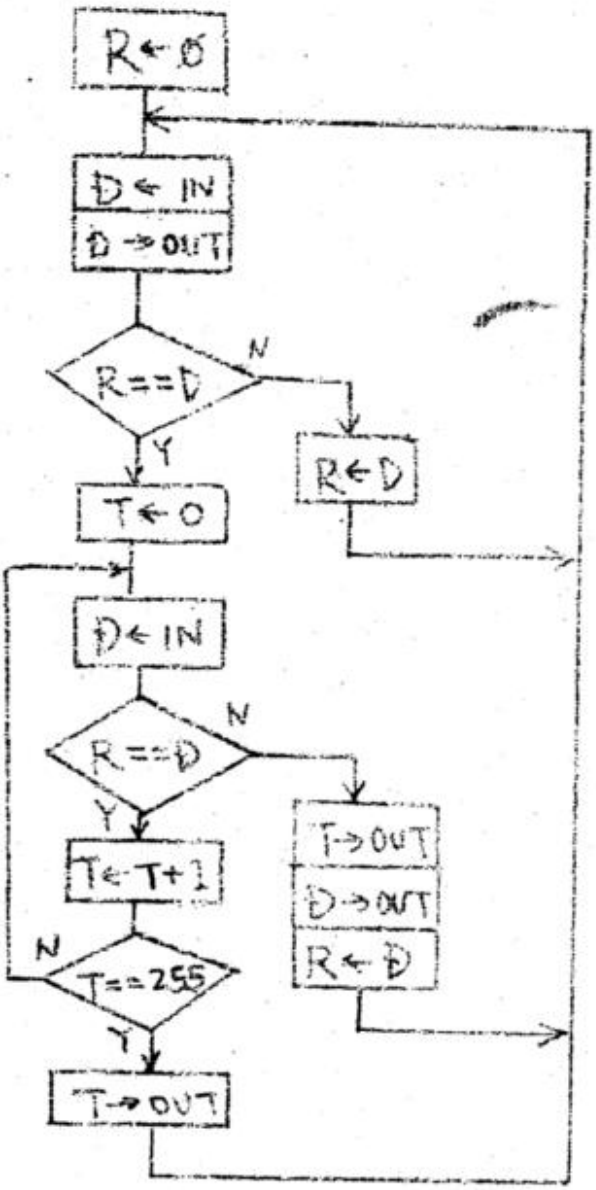


図 11 （圧縮）

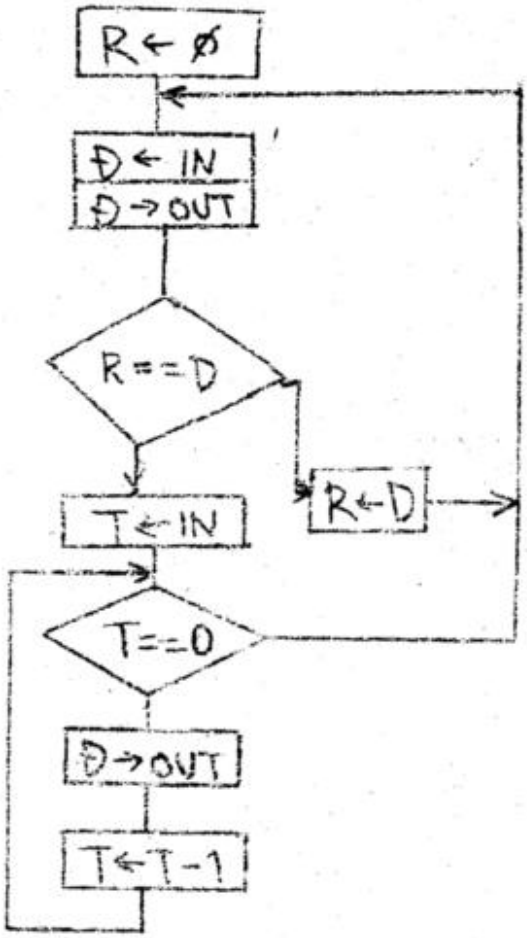


図 12 (解凍)

図11で、D←IN はソースデータから1バイトとってきて代入すること、D→OUT は圧縮データを入れるファイルに1バイト出力することをそれぞれ表し、
図12では、D←IN は圧縮ファイルから1バイト読んできて、D→OUT は、RS-232 への出力をそれぞれ表します。(なお、簡単のため、終了のチェックは省いて描いてあります。)

98側でやっていることは以上で全て終わりです。
あとはRS-232Cから送られてくるデータをMSX2で受けとって、V-RAMへそのまま送り画面を表示させた後リレーのON-OFFを制御します。このMSX2側の処理は全てマシン語でプログラムを組みました。

MSX2に関してはアセンブラやエディタ他のツールがいっさい手持ちがなかったために簡易アセンブラをBASICで組んで使いました。この簡易アセンブラプログラムについては、ニーモニック(Z-80ザイログ)とマシン語コードとの変換ルールをデータの形でファイルに持たせておいてアセンブルを行ないます。そして今まで見たことないニーモニックに出会うと、対話形式で変換ルールを聞いてきます。それを入力してやると、最後のデータをファイルにアペンドして、アセンブルを続行するという手法のプログラムにしました。

ですからこのアセンブラプログラムは最初な頭がからっぽで、いろいろたずねてきますが、それに答えて教育をちゃんとしてやるとだんだんかしこくなっていくとゆーなかなかおもしろいものです。

このやり方の良かった点は、自分のよく使うニーモニックが限られているような場合、結果的に検索にムダがなくなって、BASICの割には、すばやくアセンブルしてくれます。それにだんだんプログラムがかしこくなっていくので教育のしがいがあるし、何となく、かっこえ感じがします。(人工知能っぽい?)

しかし何にいってもBASICはBASICスピードがネックとなります。特に、教育が過ぎて余分なデータがいっぱいぶらさがってしまうと、スピードがぐぐっとのろくなってしまいます。このあたりはまだまだ改良の余地があると思います。

しかしこのプログラムのもう1つの利点は、最初は白紙からスタートするわけですから、他のCPUのニーモニックについて教育してやれば、そのCPUのアセンブラができてしまいます。つまり汎用性があるプログラムと言えるでしょう。

好きかってなことをいろいろつづってきましたが、このへんで、そろそろ終わります。約半年間にわたっていろいろ試行錯誤のあげくに、なんとか、30秒程度のアニメーションが完成しました。そのドキュメントをするには、あまりに乱雑な内容ですが、少しでもその足跡が残せれば幸いです。また本コンピュータ部の後輩諸氏にこのプログラムを基に改良、発展させてくれることを期待します。最後に乱文乱筆(+乱内容)をお許し下さい。

1989・11・21(火)

KIT コンピュータ部 部長 こんちゃん。

Scheme ってなあに?

小前 晋

1 はじめに

昔から、「Scheme! Scheme!」と言ってたので、今回は Scheme のお話をします。

Scheme は Lisp の 1 つの方言です。この後は Lisp の知識があるものとして書いていきます。途中で Scheme のプログラムが出て来るので、そのために少しお勉強しておきましょう。

Scheme では、副作用の起こり得るものには、"!"(「バン」と発音する)が付き、真偽を問うものには、"?"(「プ」又は「ピー」と発音する)が付きます。[1]
例えば、

一般的な Lisp	Scheme
setq	set!
rplaca	set-car!
rplacd	set-cdr!
eq	eq?
equal	equal?
zerop	zero?
symbolp	symbol?

また真偽値は、真を'#t 、偽を #f で表します。

[1]ただし、=, <, >, <=, >= は例外

関数定義の方法の違いは…

- Lisp

```
(defun fact (n)
  (if (zerop n)
      n
      (* (fact (1- n)) n)))
```

- Scheme

```
(define (fact n)
  (if (zero? n)
      n
      (* (fact (1- n)) n)))
```

現在、「 Revised^3 Report on the Algorithmic Language Scheme 」に沿って作られたものとして、

- PC Scheme
- Mac Scheme
- C Scheme

などがあります。

Scheme の主な特徴には

1. 型の区別がない。
2. 関数は値呼び出し。
3. 静的スコープ (static scope, lexical scope) を採用している。
4. tail-recursive な実装をしている。
5. 関数 (function) が first class object として扱える。

6. 継続 (continuation) が first class object として扱える。

などがあります。

1 は Lisp ならあたりまえで、2 は最近の Lisp ではあたりまえ (?) なので、以下では、3 、4 、5 、6 について説明していきます。

2 静的スコープ (static scope, lexical scope) を採用

静的スコープは、簡単に言えば Pascal と同じスコープです。

旧来の Lisp は、動的スコープ (dynamic scope) が主流でした。なぜなら、動的スコープは interpreter に適しているからです。変数のバインド情報を A-list で管理しておき、関数の呼び出し毎にこの A-list にローカルなバインド情報を付け加えるだけですから。

しかし、最近の Lisp は静的スコープが主流です。これは、最近の Lisp が compiler を備えるようになったからです。つまり、compile したコードを実行する時、lexical に決定できない変数をアクセスするには、一度テーブルか何かを引かなければいけなくて、テーブルを引くなら interpreter とそれほど違わないので、速度向上があまり望めない。interpreter と compiler で動作が違うのは論外だし、compile したものが遅いのでは complier のうまみがない。そこで、静的スコープというわけです。

3 tail-recursive な実装

Scheme では tail-recursive な実装をしているために、不要のスタックを消費しません。具体的に、例を挙げて説明しましょう。

Basic で、

```
10 GOSUB 200
20 END

100 PRINT "XYZ"
```

```
110 RETURN

200 PRINT "ABC"
210 GOSUB 100
220 RETURN
```

というプログラムがあったとすると、

```
10 GOSUB 200
20 END

100 PRINT "XYZ"
110 RETURN

200 PRINT "ABC"
210 GOTO 100
```

と変えた方が速くて、ネストが 1 レベル深くならずに済むのはわかりますね。これと同じことをユーザの見えない所でやってしまうことを tail-recursive な実装をしているといいます。

例えば、

```
(define (even? n)
  (if (zero? n) #t (odd? (1- n))))
(define (odd? n)
  (if (zero? n) #f (even? (1- n))))
```

は、互いに呼び合うが、スタックは消費しないので、大きな数を引数にしても平気です。

4 関数 (function) が first class object

関数が first class object ということは、関数も数とか記号とかと同様に扱えるということです。

Scheme では、関数名を表す記号の値はその関数自身なので、

```
> (set! foo car)
...
> (foo '(a b c))
a
```

ということができ、

```
> ((if #t + *) 2 3)
5
> ((if #f + *) 2 3)
6
```

ということもできます。

関数が first class object だから、関数を引数とするようなメタな関数も簡単に作れます。
例えば、与えられた関数に同じ引数を二つ作用させるような関数を作る関数は、

```
> (define ((duplicate f) n) (f n n))
...
> (define square (double *))
...
> (square 5)
25
> ((duplicate +) 3)
6
```

のようになります。

関数定義の方法として、define を紹介しましたが、本来 define というのは単に新しい変数を生成するもので、関数を定義するには、

```
(define add4 (lambda (x) (+ x 4)))
```

としなければなりません。先に紹介した方法は、頻繁に使われる関数定義のための簡略形なのです。

5 継続 (continuation) が first class object

continutation とは、プログラムの実行途中の状態を指します。当然、他の言語にも continuation は概念としてありますが、Scheme では、これもユーザが自由に扱えるようになっています。

つまり、ユーザが自由にプログラムの実行途中の状態をつかみ、その状態に飛び込むことができるのです。

まず、つかむには、

```
(call-with-current-continuation 〈1 引数の関数〉)
```

とすればよくて、〈1 引数の関数〉の引数にこの式の continuation がバインドされます。(この後では call-with-current-continuation を略して call/cc とします。)

そして飛び込むには、continuation を 1 引数の関数のように使います。引数は、飛び込んできた時の

```
(call/cc 〈1 引数の関数〉)
```

自体の値になります。

例えば、

```
(define (list-length obj)
  (call/cc
    (lambda (cont)
      (define (r obj)
        (cond ((null? obj) 0)
              ((pair? obj)
                (1+ (r (cdr obj))))
              (else (cont #f))))
      (r obj))))

> (list-length '(1 2 3 4))
```

```
4
> (list-length '(a b . c))
#f
```

のようになります。

この continuation を使えば、コルーチンが簡単に表せます。
コルーチンの有名な問題である reader-writer 問題 (producer-consumer 問題) を書いてみると…

```
(define (start)
  (define *data* 0)                          ; data initialize
  (define (writer cont)
    (set! cont (call/cc (lambda (c) (set! writer c) cont)))
    (set! *data* (1+ *data*))                ; produce data
    (cont writer))
  (define (reader cont)
    (set! cont (call/cc (lambda (c) (set! reader c) cont)))
    (write *data*) (write-char #\space)      ; consume data
    (cont reader))
  (writer reader))
```

この例では、互いに呼び出していますが、tail-recursive な実装のためにスタックを消費しません。そのため、これを実行すると無限に数字を表示し続けます。

6 最後に

だから何だと言われても困るのですが、関数や、プログラムの途中の状態 (continuation) が、ただのデータとして扱えたりするというのは正しいと思うでしょ。tail-recursive な実装というのも、正しいことでしょ。(何、正しいと思わないって！それは、あなたの頭が腐っているのです。) 関数閉包や、continuation を使えば、オブジェクトオリエンティッドなプログラムの安易な実現が出来ますしね。

やはり、こういう正しい言語でプログラムが書ける環境にいることが精神衛生上よろしいようですね。もっとも私は、アセンブラで Scheme

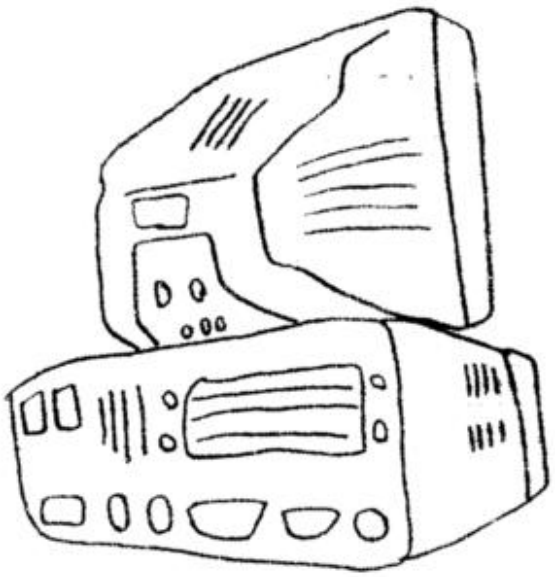
の処理系を作る毎日ですが、これが精神衛生上よろしいかどうかは、う
〜む…わからない。

あとがき

「今年のLime出版は難しいのでは？」
とゆー ウワサ にも負けず なんとか一応
今年も出すことができました。
私の不手ぎわ から印刷に手まどって
てしまい 原稿を寄せてくれた方々を
はじめ 手伝ってもらった階さんに大
変御迷惑をおかけしました。この
場をかりて お侘び申し上げます。
今後とも さらに内容の充実を計り
途断えることなく出版してゆこうと思
います。

1989・11・22
Lime 編集長
近藤 政雄

^C
Break in 1989
Ok



Lime

'89 学祭号

KIT コンピュータ部