PROGRAM LAYOUT

Most MATH users are "Topsy" programmers: The program grows at the console -- which, of course, is the idea behind MATH. But for large problems it is an inefficient use of man and machine time, and ties up a scarce resource -- consoles.

Large problems should be blocked out first at the desk. Define the main task, the subtasks generated, etc., to get the problem "tree." Indicate which subtasks must precede which others. List formulas and abbreviations. View each MATH part as being responsible for executing a subtask. Part 1 may be the overall executive. Establish input and output housekeeping formats.

VOLUME CALCULATION

```
Delete all.
Get vol-calculation.
Done.
Type all.

1.1 Type "I can compute the volume of a sphere(1), cylinder(2), or cone(3)".
1.2 Type "if you will indicate which, please.".
1.3 Demand w.
1.35 Type "How was that again?" if t.
1.36 To step 1.3 if t.
1.4 Do part 2.
1.5 Type v in form (v<10000:1;2).
1.6 To step 1.3.

2.1 Type "Radius?".
2.2 Demand r.
2.3 Done if w=1.
2.4 Type "Height?".
2.5 Demand h.

Form 1:
The volume is ___.__

Form 2:
The volume is ........

         t:   not (w=1 or w=2 or w=3)
         v:   (w=1:4/3·p·r*3;w=2:p·r*2·h;w=3:p/3·r*2·h)

         p =        3.14159265

Do part 1.
I can compute the volume of a sphere(1), cylinder(2), or cone(3)
if you will indicate which, please.
        w = 3
Radius?
        r = 3.1
Height?
        h = 5.0
The volume is  50.32
        w = 1
Radius?
        r = 34.057
The volume is  1.65 05
        w = 2
Radius?
        r = 800
Height?
        h = 10*6
The volume is  2.01 12
        w = 5
How was that again?
        w =
```

PRIME NUMBERS

```
Delete all.
Get prime-number.
Done.
Type all.

1 Type x if P(x).
```

The function P(x) has value *true* *if and only if x is prime. After P(x) filters out invalid cases its subfunction p(x) finds the first exact divisor of x--via fp(x/i)=0 and compares that divisor with x. Then y is prime if the first exact divisor is x itself.*

```
2 Type x in form 1+tv(P(x)) if (x=true or x= false:true;fp(x)=0).

Form 1:
_____________ is not prime.

Form 2:
_____________ is prime.

        P(x):  (x=true or x=false:false;x≤1 or fp(x)≠0:false;p(x))
        p(x):  first[i=2,3(2)[x<10:3;ip(sqrt(x))],x:fp(x/i)=0] =x

          x =        54

Do part 2 for x=1, 2, -5, .03, true, false, 17.
          1 is not prime.
          2 is prime.
         -5 is not prime.
       true is not prime.
      false is not prime.
         17 is prime.
Do part 1 for x=1(1)30.
          x =         2
          x =         3
          x =         5
          x =         7
          x =        11
          x =        13
          x =        17
          x =        19
          x =        23
          x =        29
```

COMPUTING EFFICIENCY--I

```
Type all.

1.1 Set m=M.
1.2 Set g=first(x=0(a)10:f(x)=m).
1.3 Type m,g.

2.1 Type M,G.

4.1 Line.
4.2 Reset timer.
4.3 Do part k.
4.4 Type k,timer in form 1.

Form 1:
Minutes for case _:  __.__

Let G = first(x=0(a)10:f(x)=M).
Let M = min(x=0(a)10:f(x)).
Let f(x) = x*3-10·x*2-6·x+10.

a=.1
Do part 4 for k=1,2.

          m =      -179
          g =         7
Minutes for case 1:    .34

          M =      -179
          G =         7
Minutes for case 2:  13.39

Type users.
     users: 13
```

This example gives some idea of economy in computing the first value of a function where a minimum is achieved. Part 1 (k=1) computes the minimum once and stores it. Part 2 (k=2) computes the minimum anew for each value of x. Here timer shows only relative efficiency since elapsed time depends on the number of users computing.

Note that if we Let G(z)=first (x=0(a)10:f(x)=z). and Type G(M), then M will only be calculated once.

COMPUTING EFFICIENCY--II

```
Delete all.
Get time-base.
Done.
Type all.

4.1 Line.
4.2 Reset timer.
4.3 Do part k.
4.4 Type k,timer in form 1.

Form 1:
Minutes for case _:  __.__

1.1 Do part 10 for x=-200(1)200.
10.1 Set d=-f(x-1)+f(x).
2.1 Do part 20 for x=-200(1)200.
20.1 Set D=f(x)-f(x-1).
30.1 Set f(i)=i*2.
Do part 30 for i=-250(1)250.

Do part 4 for k=1,2.

Minutes for case 1:    .29

Minutes for case 2:    .69
```

This example illustrates, in part 10 and part 20, two equivalent ways of computing (but not displaying) successive differences of elements in the vector f. The program in part 4 tries both forward and backward methods of proceeding through f and compares time of computation for the two.

Since MATH assigns storage only for array elements that have an assigned value, each time an array is referenced by the program a search for the requested array element must be made. MATH facilitates forward (increasing index values) searches by remembering the index of the last reference made and starting each new search from that point.

COMPUTING EFFICIENCY--III

Numeric literals given in MATH computations require time for conversion to internal form in proportion to their length in characters. In case 2 at the left, the number 1.23456789 is converted for each value of i in the summation, while in case 1 only one conversion was required when a was input.

```
Delete all.
Get time-base.
Done.
Type all.

4.1 Line.
4.2 Reset timer.
4.3 Do part k.
4.4 Type k,timer in form 1.

Form 1:
Minutes for case _: __.__

1.1 Type sum(i=1(1)n: a+i ).
2.1 Type sum(i=1(1)n: 1.23456789+i).
n=1000
a=1.23456789
Do part 4 for k=1,2.

sum(i=1(1)n: a+i ) = 501734.817
Minutes for case 1:    .38

sum(i=1(1)n: 1.23456789+i) = 501734.817
Minutes for case 2:    .47
```

DEGREE-RADIAN CONVERSION FORMULAS

```
A stored program displays the formulae and sample usage. *
Do part 1.
r:degrees to radians
h:time to radians
d:radians to degrees
t:radians to time
s:sum of two angles

     D(r,c):  ip(r·c)+.01·ip[60·fp(r·c)]+.006·fp(60·r·c)
     R(d,c):  [ip(d)+ip(100·fp[d])/60+fp(100·d)/36]/c
       d(x):  D(x,45/arg(1,1))
       h(x):  R(x,3/arg(1,1))
       r(x):  R(x,45/arg(1,1))
     s(x,y):  d(r(x)+r(y))
       t(x):  D(x,3/arg(1,1))

Some examples:

    r(45.1753) =          .790600213
  d(.790600213) =       45.1753
    h(13.4739) =          3.61130437
  t(3.61130437) =       13.4739
  t(r(23.2951)) =        1.33594
  d(h(7.0205)) =       105.3115
  s(11.4937,26.3351) = 38.2328
  s(21.4703,-39.5702) = -18.0959
```

BOOLEAN FUNCTIONS

```
Type all parts, all forms.

1 Page if $≠1.
1.1 Type all formulas,form 1,_.
1.2 Do part 2 for x=true,false.
1.3 Page.

2 Do part 3 for y=true,false.

3 Type x,y,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p in form 2.


Form 1:
   x      y     a b c d e f g h i j k l m n o p

Form 2:
______ ______  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

This program displays the values of the 16 boolean functions of two variables.

```
Do part 1.

          a:  |x|
          b:  |y|
          c:  |x=y|
          d:  |x≠y|
          e:  |true|
          f:  |false|
          g:  |not x|
          h:  |not y|
          i:  |x or y|
          j:  |x and y|
          k:  |not x or y|
          l:  |not y or x|
          m:  |not(x or y)|
          n:  |not x and y|
          o:  |not y and x|
          p:  |not(x and y)|
```

The 16 boolean functions.

x	y	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
true	true	1	1	1	0	1	0	0	0	1	1	1	1	0	0	0	0
true	false	1	0	0	1	1	0	0	1	1	0	0	1	0	0	1	1
false	true	0	1	0	1	1	0	1	0	1	0	1	0	0	1	0	1
false	false	0	0	1	0	1	0	1	1	0	0	1	1	1	0	0	1

Values of the 16 functions.

PROGRAM CHAINING

Master program in chain.

```
Get master-program.
Done.
Type all.

1.05 Type "Fetching the vector f from the files.".
1.1 Get data-f.
1.2 Do part 2 for i=1(1)100.
1.3 Delete f.
1.35 Type "Fetching the vector g from the files.".
1.4 Get data-g.
1.5 Do part 3 for i=1(1)100.
1.54 Store g replacing data-g.
1.6 Type "New values for g have been computed and filed.".
```

Data chain.

Compute moving averages.

```
2.1 Set s(i)=sum(k=i-10(1)i+10:f(i) )/21.

3.1 Set g(i)=g(i)+s(i).

10.05 Do part 1.
10.1 Delete part 1, part 2, part 3.
10.15 Type "Getting a second processing program from the files.".
10.2 Get program(2).
10.3 Do part 1.
```

Auxiliary program.

```
Delete all.
Get program(2).
Done.
Type all.

1.1 Type sum(i=1(1)100:g(i)).
```

Sample execution.

```
Delete all.
Get master-program.
Done.
Do part 10.
Fetching the vector f from the files.
Fetching the vector g from the files.
New values for g have been computed and filed.
Getting a second processing program from the files.
sum(i=1(1)100:g(i)) = 500
```

ROOT FINDING

```
Delete all.
Get root-finder.
Done.
Type all.

1 Type r(x),p(r(x)) in form 1.

Form 1:
x =  __.______          p(x) =   _.______

          i(x):   x-p(x)/q(x)
          p(x):   x*3-10•x*2-6•x+10
          q(x):   [p(x+d)-p(x)]/d
          r(x):   [|p(x)|<10*(-6):x;r(i(x))]

             d =          1•10*(-4)

Do part 1 for x = -10,1,10.
x = -1.24670       p(x) =    .00000
x =    .76528      p(x) =    .00000
x = 10.48142       p(x) =    .00000
```

Find the roots of the polynomial p(x) by Newton's method expressed as i(x), where q(x) is the approximate derivative of p(x); r(x) recursively improves the root until a value sufficiently close to zero is obtained.

GAUSSIAN INTEGRATION

```
Get gauss-integrate.
Done.
Type all.


        I(f):   h/2•sum(i=1(1)n: sum[j=1(1)m: f(x(i,j))])
           h:   (b-a)/n
      x(i,j):   a+h/2•[t(j)+2•i-1]

           m =          2
           n =         30

a=0
b=1
t(1) = 1/sqrt(3)
t(2) = -t(1)
Type exp(1)-1, I(exp).
    exp(1)-1 =          1.71828183
      I(exp) =          1.71828184
```

PROBABILITY INTEGRAL

This example illustrates a complete MATH program, including commands to control output formatting, and the results of its execution.

```
Type all.

1.13 Type form 2.
1.15 Do part 2 for b=.1(.1)4.

2.05 Set a=-b.
2.1 Line if fp($/5)=1/5.
2.6 Type b,exp(b),log(exp(b)),c•I(f) in form 1.

Form 1:
__.___   .......   __.____   _.____

Form 2:
  X      EXP(X)      LOG       PROB

     I(f):   h/2•sum(i=1(1)30:sum[j=1(1)2:f(y(i,j))])
   Y(i,j):   a+h/2•[t(j)+2•i-1]
     f(x):   exp(-x*2/2)
        h:   (b-a)/30
   y(i,j):   a+h/2•[t(j)+2•i-1]

         a =          -.1
         b =           .1
         c =           .398942281

      t(1) =           .577350268
      t(2) =          -.577350268

Do part 1.
   X      EXP(X)      LOG       PROB

  .10     1.1 00     .100      .0797
  .20     1.2 00     .200      .1585
  .30     1.3 00     .300      .2358
  .40     1.5 00     .400      .3108

  .50     1.6 00     .500      .3829
  .60     1.8 00     .600      .4515
  .70     2.0 00     .700      .5161
  .80     2.2 00     .800      .5763

  .90     2.5 00     .900      .6319
 1.00     2.7 00    1.000      .6827
 1.10     3.0 00    1.100      .7287
 1.20     3.3 00    1.200      .7699

 1.30     3.7 00    1.300      .8064
 1.40     4.1 00    1.400      .8385
 1.50     4.5 00    1.500      .8664
 1.60     5.0 00    1.600      .8904

 1.70     5.5 00    1.700      .9109
 1.80     6.0 00    1.800      .9281
 1.90     6.7 00    1.900      .9426
```