# A Torture Test for METAFONT

by Donald E. Knuth
Stanford University

(*Version 2, January 1990*)

Programs that claim to be implementations of METAFONT84 are supposed to be able to process the test routine contained in this report, producing the outputs contained in this report.

The preparation of this report was supported in part by the National Science Foundation under grants IST-8201926 and MCS-8300984, and by the System Development Foundation. 

**Introduction.** People often think that their programs are "debugged" when large applications have been run successfully. But system programmers know that a typical large application tends to use at most about 50 per cent of the instructions in a typical compiler. Although the other half of the code—which tends to be the "harder half"—might be riddled with errors, the system seems to be working quite impressively until an unusual case shows up on the next day. And on the following day another error manifests itself, and so on; months or years go by before certain parts of the compiler are even activated, much less tested in combination with other portions of the system, if user applications provide the only tests.

How then shall we go about testing a compiler? Ideally we would like to have a formal proof of correctness, certified by a computer. This would give us a lot of confidence, although of course the formal verification program might itself be incorrect. A more serious drawback of automatic verification is that the formal specifications of the compiler are likely to be wrong, since they aren't much easier to write than the compiler itself. Alternatively, we can substitute an informal proof of correctness: The programmer writes his or her code in a structured manner and checks that appropriate relations remain invariant, etc. This helps greatly to reduce errors, but it cannot be expected to remove them completely; the task of checking a large system is sufficiently formidable that human beings cannot do it without making at least a few slips here and there.

Thus, we have seen that test programs are unsatisfactory if they are simply large user applications; yet some sort of test program is needed because proofs of correctness aren't adequate either. People have proposed schemes for constructing test data automatically from a program text, but such approaches run the risk of circularity, since they cannot assume that a given program has the right structure.

I have been having good luck with a somewhat different approach, first used in 1960 to debug an ALGOL compiler. The idea is to construct a test file that is about as different from a typical user application as could be imagined. Instead of testing things that people normally want to do, the file tests complicated things that people would never dare to think of, and it embeds these complexities in still more arcane constructions. Instead of trying to make the compiler do the right thing, the goal is to make it fail (until the bugs have all been found).

To write such a fiendish test routine, one simply gets into a nasty frame of mind and tries to do everything in the unexpected way. Parameters that are normally positive are set negative or zero; borderline cases are pushed to the limit; deliberate errors are made in hopes that the compiler will not be able to recover properly from them.

A user's application tends to exercise 50% of a compiler's logic, but my first fiendish tests tend to improve this to about 90%. As the next step I generally make use of frequency-counting software to identify the instructions that have still not been called upon. Then I add ever more fiendishness to the test routine, until more than 99% of the code has been used at least once. (The remaining bits are things that can occur only if the source program is really huge, or if certain fatal errors are detected; or they are cases so similar to other well-tested things that there can be little doubt of their validity.)

Of course, this is not guaranteed to work. But my experience in 1960 was that only two bugs were ever found in that ALGOL compiler after it correctly translated that original fiendish test. And one of those bugs was actually present in the results of the test; I simply had failed to notice that the output was incorrect. Similar experiences occurred later during the 60s and 70s, with respect to a few assemblers, compilers, and simulators that I wrote.

This method of debugging, combined with the methodology of structured programming and informal proofs (otherwise known as careful desk checking), leads to greater reliability of production software than any other method I know. Therefore I have used it in developing METAFONT84, and the main bulk of this report is simply a presentation of the test program that was used to get the bugs out of METAFONT.

Such a test file is useful also after a program has been debugged, since it can be used to give some assurance that subsequent modifications don't mess things up.

The test file is called `TRAP.MF`, because of my warped sense of humor: METAFONT's companion system, TEX, has a similar test file called `TRIP`, and I couldn't help thinking about Billy Goat Gruff and the story of "trip, trap, trip, trap."

The contents of this test file are so remote from what people actually do with METAFONT, I feel apologetic if I have to explain the correct translation of `TRAP.MF`; nobody really cares about most of the nitty-

gritty rules that are involved. Yet I believe TRAP exemplifies the sort of test program that has outstanding diagnostic ability, as explained above.

If somebody claims to have a correct implementation of METAFONT, I will not believe it until I see that TRAP.MF is translated properly. I propose, in fact, that a program must meet two criteria before it can justifiably be called METAFONT: (1) The person who wrote it must be happy with the way it works at his or her installation; and (2) the program must produce the correct results from TRAP.MF.

METAFONT is in the public domain, and its algorithms are published; I've done this since I do not want to discourage its use by placing proprietary restrictions on the software. However, I don't want faulty imitations to masquerade as METAFONT processors, since users want METAFONT to produce identical results on different machines. Hence I am planning to do whatever I can to suppress any systems that call themselves METAFONT without meeting conditions (1) and (2). I have copyrighted the programs so that I have some chance to forbid unauthorized copies; I explicitly authorize copying of correct METAFONT implementations, and not of incorrect ones!

The remainder of this report consists of appendices, whose contents ought to be described briefly here:

Appendix A explains in detail how to carry out a test of METAFONT, given a tape that contains copies of the other appendices.

Appendix B is TRAP.MF, the fiendish test file that has already been mentioned. People who think that they understand METAFONT are challenged to see if they know what METAFONT is supposed to do with this file. People who know only a little about METAFONT might still find it interesting to study Appendix B, just to get some insights into the methodology advocated here.

Appendix C is TRAPIN.LOG, a correct transcript file TRAP.LOG that results if INIMF is applied to TRAP.MF. (INIMF is the name of a version of METAFONT that does certain initializations; this run of INIMF also creates a binary base file called TRAP.BASE.)

Appendix D is a correct transcript file TRAP.LOG that results if INIMF or any other version of METAFONT is applied to TRAP.MF with base file TRAP.BASE.

Appendix E is TRAP.TYP, the symbolic version of a correct output file TRAP.72270GF that was produced at the same time as the TRAP.LOG file of Appendix D.

Appendix F is TRAP.PL, the symbolic version of a correct output file TRAP.TFM that was produced at the same time as the TRAP.LOG file of Appendix D.

Appendix G is TRAP.FOT, an abbreviated version of Appendix D that appears on the user's terminal during the run that produces TRAP.LOG, TRAP.72270GF, and TRAP.TFM.

The debugging of METAFONT and the testing of the adequacy of TRAP.MF could not have been done nearly as well as reported here except for the magnificent software support provided by my colleague David R. Fuchs. In particular, he extended our local Pascal compiler so that frequency counting and a number of other important features were added to its online debugging abilities.

The method of testing advocated here has one chief difficulty that deserves comment: I had to verify by hand that METAFONT did the right things to TRAP.MF. This took many hours, and perhaps I have missed something (as I did in 1960); I must confess that I have not checked every single number in Appendices D, E, and F. However, I'm willing to pay $81.92 to the first finder of any remaining bug in METAFONT, and I will be surprised if that bug doesn't show up also in one of these appendices.

**Appendix A: How to test METAFONT.**

0. Let's assume that you have a tape containing `TRAP.MF`, `TRAPIN.LOG`, `TRAP.LOG`, `TRAP.TYP`, `TRAP.PL`, and `TRAP.FOT`, as in Appendices B, C, D, E, F, and G. Furthermore, let's suppose that you have a working `WEB` system, and that you have working programs `TFtoPL` and `GFtype`, as described in the TEXware and METAFONTware reports.
1. Prepare a version of `INIMF`. (This means that your `WEB` change file should have **init** and **tini** defined to be null.) The **debug** and **gubed** macros should be null, in order to activate special printouts that occur when *tracingedges* $> 1.0$. The **stat** and **tats** macros should also be null, so that statistics are kept. Set *mem_top* and *mem_max* to 3000 (or to *mem_min* plus 3000, if *mem_min* isn't zero), for purposes of this test version. Also set *error_line* $= 64$, *half_error_line* $= 32$, *max_print_line* $= 72$, *screen_width* $= 100$, and *screen_depth* $= 200$; these parameters affect many of the lines of the test output, so your job will be much easier if you use the same settings that were used to produce Appendix E. Also (if possible) set *gf_buf_size* $= 8$, since this tests more parts of the program. You probably should also use the "normal" settings of other parameters found in `MF.WEB` (e.g., *max_internal* $= 100$, *buf_size* $= 500$, etc.), since these show up in a few lines of the test output. Finally, change METAFONT's screen-display routines by putting the following simple lines in the change file:

```
@x Screen routines:
begin init_screen:=false;
@y
begin init_screen:=true; {screen instructions will be logged}
@z
```

   None of the other screen routines (*update_screen*, *blank_rectangle*, *paint_row*) should be changed in any way; the effect will be to have METAFONT's actions recorded in the transcript files instead of on the screen, in a machine-independent way.
2. Run the `INIMF` prepared in step 1. In response to the first '`**`' prompt, type carriage return (thus getting another '`**`'). Then type '`\input trap`'. You should get an output that matches the file `TRAPIN.LOG` (Appendix C). Don't be alarmed by the error messages that you see, unless they are different from those in Appendix C.
3. Run `INIMF` again. This time type '␣`&trap`␣␣`trap`␣'. (The spaces in this input help to check certain parts of METAFONT that aren't otherwise used.) You should get outputs `TRAP.LOG`, `TRAP.72270GF`, and `TRAP.TFM`. Furthermore, your terminal should receive output that matches `TRAP.FOT` (Appendix G). During the middle part of this test, however, the terminal will not be getting output, because `batchmode` is being tested; don't worry if nothing seems to be happening for a while—nothing is supposed to.
4. Compare the `TRAP.LOG` file from step 3 with the "master" `TRAP.LOG` file of step 0. (Let's hope you put that master file in a safe place so that it wouldn't be clobbered.) There should be perfect agreement between these files except in the following respects:
   a) The dates and possibly the file names will naturally be different.
   b) If you had different values for *stack_size*, *buf_size*, etc., the corresponding capacity values will be different when they are printed out at the end.
   c) Help messages may be different; indeed, the author encourages non-English help messages in versions of METAFONT for people who don't understand English as well as some other language.
   d) The total number and length of strings at the end and/or "still untouched" may well be different.
   e) If your METAFONT uses a different memory allocation or packing scheme, the memory usage statistics may change.
   f) If you use a different storage allocation scheme, the capsule numbers will probably be different, but the order of variables should be unchanged when dependent variables are shown. METAFONT should also choose the same variables to be dependent.
   g) If your computer handles integer division of negative operands in a nonstandard way, you may get results that are rounded differently. Although TEX is careful to be machine-independent in this regard, METAFONT is not, because integer divisions are present in so many places.

5. Use GFtype to convert your file TRAP.72270GF to a file TRAP.TYP. (Both of GFtype's options, i.e., mnemonic output and image output, should be enabled so that you get the maximum amount of output.) The resulting file should agree with the master TRAP.TYP file of step 0, assuming that your GFtype has the "normal" values of compile-time constants (*top_pixel* = 69, etc.).
6. Use TFtoPL to convert your file TRAP.TFM to a file TRAP.PL. The resulting file should agree with the master TRAP.PL file of step 0.
7. You might also wish to test TRAP with other versions of METAFONT (i.e., VIRMF or a production version with another base file preloaded). It should work unless METAFONT's primitives have been redefined in the base file. However, this step isn't essential, since all the code of VIRMF appears in INIMF; you probably won't catch any more errors this way, unless they would already become obvious from normal use of the system.

**Appendix B: The TRAP.MF file.** The contents of the test routine are prefixed here with line numbers, for ease in comparing this file with the error messages printed later; the line numbers aren't actually present.

```
% This is a diabolical test file for MF84. Don't get stuck.
if not known inimf: .inimf.=.0. % next lines are skipped if format loaded
  inimf; nonstopmode; tracingtitles:=day; showstopping:=year; hppp:=1000;
  << == >> ::: || `` '' -- !! ?? ## && @@ $$ [[ ]] {{ }} . (( 5.5.5 )) ++ "..";
  begingroup save =; let=,; save,; newinternal $=,; let ):=, endgroup;
  let year=month; showvariable errorstopmode,readstring,2,"2",,,(,),<<,year;
  tracingrestores:=tracingcommands:=.00000762939453125; % that's 2^(-17)
  if tracingcommands>0:tracingcommands:=if not cycle "":1.1 forever;fi;
  tracingcommands:=2.1 exitif tracingcommands>2 endfor; showtoken |=:|>;
  tracingedges:=1/.00001; tracingequations:=$+1; p~=tracingedges+.00001;
  interim tracingspecs:=tracingpens:=tracingchoices:=tracingstats:=
  warningcheck:=tracingoutput:=tracingmacros:=1; $:=ASCII""; $:=x; p~:=p~;
  delimiters (); delimiters begintext endtext; vardef foo(text t)=t enddef;
  def lig(text t,|)=ligtable0::for *=1step1until60:0kern boundarychar+*,endfor
  skipto0;ligtable t:t|0,skipto255;boundarychar:=boundarychar+51.29999enddef;
  foo begintext interim proofing:=(-.5; shipout nullpicture; special"3" endtext;
  for n=tracingpens step 1 until proofing:fi endfor showstats; let!!=skipto;
  path p~; p~=(0,0)..controls (15,4) and (-15,-12)..(4,0);  everyjob /*\;;
  vardef /*\``@#=message @ & str#@ &jobname&char ASCII``&str@#!enddef;
  let next=dump; vardef ``=`` enddef; def ``="\*/" enddef;
elseif known"": ``pass[2.]; outer\; let next=\; delimiters ^^7! fi
next\; % the second pass will now compute silently; the first pass will halt
batchmode; ^^7,endgroup pausing:=1; exitif p exitif boolean pen pencircle endfor
scantokens begingroup message char0&"watch this"; "pair p[],';"&char-1endgroup
path p[][]p,w,qw; qw=(1,-2)..(2,-1)..(2.5,0.5)..(1,2)..(turningnumber',2.5);
numeric p[][]; p[[ [$] ]]=10000000000000000; "this string constant is incomplete
string foo[]p,p~if true:[];
boolean p[]~,boolean fi.boolean; showvariable boolean; def\\= =end enddef;
picture e[]e[], e[], p~[]~[][];
pen p~[]~,q["a",qq; p~1~=q=pencircle scaled mexp(-3016.57654);
transform p,pp0; if p=p:qq=makepen((1,0)..cycle) xscaled hex "1000";fi
qq:=makepen((0,0)..(1,0)..(0,1)..(0,0)..(1,0)..(0,1)..cycle);
vardef p[][]p~ begintext suffix a,b endtext())suffix@=show #@; p.a.b() enddef;
expandafter let\endtext\\; outer endtext,\,;;qq:=pencircle scaled 4.5 yscaled 2;
(6,12)-p7=(0,1)transformed p=(2/(x-x),3/0)transformed p;
p1\2p=p007=begintext if string p~[$]: p.1.2-p.1.199999,1 endtext transformed p;
showstopping:=0;showvariable p; p=p; let [[=begingroup; let ]]=endgroup;
(xxpart p+.002,yxpart p)=1[p1,p2]=(5,y+.00001)=(5,y)=(yypart p,xypart p);
reverse(p~..cycle) transformed p=p2. 3.p;
p[000000000001]2p~(,[2]3p~,-)=p~1~2[pausing];
vardef p~[]@# tertiary t:=if p@ @=@ @p fi; vardef p[][]p~[]=BAD; inner ;;
show p~[-2]~[3000,x]++4000>path p3; showvariable p,P;
numeric p[]~; p3~=2alpha; p[1/$]~=3beta;
begingroup save p; showvariable p; 3beta=1]]; showvariable p;
def//expr;=enddef;def!primary!false):!fi enddef;
def _aa__ secondary _a_=if(true enddef; qq:=makepen(qw..(qw scaled$)..cycle);
primarydef _**__=[[show _*__]] enddef;
secondarydef _***__=expandafter __ scantokens"**oct"_ enddef;
// //pencircle slanted length p~**makepen reverse subpath p7-p2 of
 (p7{p2}..controls _aa__ not odd.1(15) and known p or !p2and-p2..{1,1}(-p2
 {curl- +1)..tension atleast1..cycle sqrt2++sqrt2***[[]];
```

```
[[interim proofing:=charcode:=-20.5;chardp:=-2048;shipout nullpicture]];
if charexists -275.50002>known p0 0p=known p~: randomseed charcode; fi
randomseed:="goof"; a[($,18++1+-+18),(2,3)]=b[(3,2),(1,$);
show (^+1,~+2) slanted-1 yscaled-2 zscaled-(3,4), p transformed(pp xscaled 9),
 pp shifted (1,2) transformed(p transformed p), -_[0][1,2]; show
floor sind mlog sqrt mexp200cosd angle(normaldeviate,uniformdeviate-chardp);
string s[]; s1=s2=s4; s3=s5; s1=s2; if s1<=s4<>(s1<>s3):show[[char34=s2:=s3]]fi;
substring penoffset point.1of.p~of[[pencircle]]rotated1080/2/1/3of decimal
 directiontime postcontrol-1.5of(p~&cycle)-precontrol1/2of p~(p~)=s1:=s4:=s4;
path p~[]; p~1=p2{length"  "}&cycle; p~1=p2=p~0; p2..controls-p2..cycle=p~2;(p7
..tension1.2..p~[length p~2]..p~2&{0,1}p2..tension1and atleast1..cycle)..tension
 x..{curl1}-p7{curl hex "IsBad"}..tension.75and.74999..p2{0,1}&p2{_,'}..cycle:=p
 ~4; subpath(-9,9)of subpath(3.5001,7.00001)of p~4=p~6;
show p~6, directiontime(1,2) of p~6, directiontime(1,-1.00001) of p~6;
p~3=(0,0)..controls (1,1) and (0,1)..(1,0); show p2..p2{p7}&{$,$}cycle,
 (directiontime(1,1) of subpath(.314159,1) of p~3)[.314159,1];
p~5=(subpath(0,.25)of p~3&subpath(.25,1)of p~3)shifted begintext1,0;
p~3:=2/3'zscaled'{p~3}..controls(2,2/3(3))and penoffset(1/2x,y)of(0,1)(1,0);
show p~3 intersectiontimes reverse p~3, point.17227 of p~3, point1-.28339of p~3;
show point xpart(p~5 intersectiontimes p~5 shifted (.01,0))of p~5-
 point ypart(p~5 intersectiontimes p~5 shifted (.01,0))of p~5;
[[interim tracingedges:=1; e[-1+-+ -1.00001]=nullpicture; addto e1 also[[
addto e0 doublepath p~5 scaled 3 withpen q; e0=e1=e2; cull e1 dropping (0,.1);
nullpicture]];show e1 shifted(4089,-4095), e2 shifted(-4095,4092)shifted (-3,0),
e2 shifted(4089,-4095)]];addto e1 also  e2 shifted(-2,$); e1:=e1 shifted(-4,$);
addto e0 also e1rotated89.999+e1scaled$; show e0 xscaled-10 yscaled2 xscaled82
yscaled683;addto e1 doublepath (0,9) withweight-3 withweight turningnumber p~6
 withpen pencircle xscaled(oct"180"++1) rotated-angle(64,$) shifted (9,8)
 withpen makepen(($,$)..(1,0)..(1,1)..($,0)..($,$)&cycle)xscaled4095.49999;
show e1, totalweight e1; chardp:=charcode:=5; xoffset:=-1.5; shipout e1;
showstats; addto e2doublepath p~ yscaled1/60; e3:=e2 yscaled 0;
autorounding:=2; addto e3doublepath(.5,0)..(3.5,1.5)withweight2;
tracingspecs:=0; q:=makepen((1,1)..cycle) yscaled 1.5;
p~8=(($,yy)rotated p{0,1}..{0,$}(1,0){0,$}..cycle)scaled2shifted(1000.49,9);
turningcheck:=1;addto e2doublepath p~8 withpen q withweight p withpen cycle p;
[[interim autorounding:=xx=.1; addto e2 contour p~8 withpen q withweight2]];
chardx:=chardp:=charext:=-14.5;shipout-(-e0-e2)+e2shifted(0,6turningnumber p~8);
p~9=(0,0)..(1,.5)..(5,1.5)..(7,2.5)..(12,3.5)..(13,4);addto e3 doublepath p~9;
smoothing:=1; addto e3 doublepath p~9; addto e3 doublepath (-4095,0)..tension
3/4 and 999..(0,2); show e3 rotated-90, (e0+e0) rotated90$ rotated90;
if "a" if "ab">"b" elseif path reverse (3,4): >="aa":foo elseif fi "bar"
else if '-(1,yy)<': :fi else def dup text t=[[t;save enddef;def|suffix$=,$
enddef; def||tertiary p=show substring p of("a" enddef;||(2,$)&"bc");
tertiarydef x++y=[[dup showtoken x;]];def quote x expr z of y=z enddef;
 texts(quote x=(y+0)y+y)("xx",foo(x))=0]] enddef; def texts(text t,tt)expr?=
 for n:=,for n"yy":n,length if false:endfor tt,t,:if string n:forsuffixes n=
 foo1,[foo(n)],':show t,tt|(n;exitif not('<='+(?,yy)) endfor for m= :+endfor
  for m=alpha step-1.1 3$: +m endfor fi endfor enddef; show (0,0){curl2}..
 subpath(23.3,4.5)of p~9{curl3}..($,$){curl4}..cycle;numspecial p~++2+3;[[
let?=if;save if,\;def if=endinput?enddef;def texts=input enddef;texts trap ]];
dup[[def texts secondary x=primarydef y++y=x@y enddef; showtoken++;x enddef]]\;
proofing:=1;texts:="a"&"b";% strings "yy" and "ab" no longer appear in memory
texts-1.00001a1=-(a2+a3+a4+a5+a6);-(a3,a2)/.99999=-(a4+a5+[[showdependencies;
```

```
a6]],a3+a4+a5+a6)]]; 1.00001a4+1=a5+a6; alpha:=.9alpha+7; showdependencies;
a5=a6=2/3-a6; cull e1 dropping($,4a5)withweight1.5; charcode:=chardp:=27;
openwindow 3 from (0,0) to (0,0) at "whoops"; addto p; shipout p; cull p;
openwindow -.5 from ($,$) to ($,$) at (0,0); special p; numspecial "p";
openwindow 15.49999 from (0,32*1024) to (1,31*1057) at (0,0); shipout e1;
openwindow 5 from (0,length((0,0){0,0})) to (4,2) at ($,1);
openwindow 6 from (length($,0),1) to (3,10) at (-5,2);
display nullpicture; display p inwindow 3; display p inwindow 6;
display e1 inwindow 6; cull e0 dropping (0.1,4095.999999) withweight 3.5
withweight-3.5; display e0 inwindow 5.5; addto e0 also p; addto e0 contour 0;
display e0 inwindow 5.49999; addto e0 contour p~9;
display e1 inwindow 3+3; display e0e0 inwindow[[vardef e[]e=enddef;6]];
addto e0 also e1; display e0 inwindow 5; ligtable||:255|=:|>>0,skipto0;
display e1 inwindow 15; display e1 inwindow 6;
show [[interim tracingcommands:=0; lig("g")(=:|); lig("h":"i")(|=:);
lig("j")(|=:>);lig("k")(=:|>); lig("l")(|=:|>)]];
b1:=c.a[ [[let c=++;vardef b=enddef;1]] ]; ligtable"m":0=:0,skipto5;
!!; errhelp 0; errmessage "Be like Jane";
errhelp "He%%%lp%"; errmessage""; errhelp ""; errmessage "Another";
headerbyte 0; headerbyte(48.5)substring(-9,9)of"long"; for\=0:\
headerbyte 9:2a6,"q"; fontdimen 9:2a6,"q"; fontdimen 1:2048;
fontmaking:=1; extensible 5 5,"c"255.5,"d"; charlist 0:5:"a":"d";
ligtable255:255::"a"=:"b","d" kern -2048,"c":0:99.5:"e"|=:|"f",0kern';
ligtable 5:0; def clear(text x)=interim x:=$ enddef; clear(hppp); vppp=0;
[[clear(tracingmacros); clear(tracingcommands); clear(tracingoutput\;
 clear(proofing);designsize:=.99999;charcode:=ASCII char-418.5;vppp:=designsize;
 def dp expr d = charcode:=charcode+1; chardp:=d; shipout nullpicture enddef;
 dp 13; dp 12; dp 0; dp 21; dp -2; dp 17; dp 11; dp 3; charic:=-1000; dp -1;
 dp 25; dp 31; dp 19; dp 7; charwd:=256; chardy:=6; dp 23; dp 30]];
def f(suffix@@)(expr a,b)(text t)=numeric w; show a; % wipes out the old w
addto @@ contour (0,0)..(2,0)..(1,$)..(1,1)..cycle withpen qq; % strange path
addto @@ doublepath (0,0){1,1}..{2,1}(2,1) withpen qq; % carefully chosen
addto @@ doublepath(($,$){1,0}..(1,1){1,0})scaled.5 withpen nullpen;
cull @@ keeping (4,4) withweight1.5; enddef; def g(suffix$)=show $ enddef;
addto e0 contour (0,0){1,0}..{1,0}(1,0){0,$}..{0,1}cycle withpen qq;
f(e[3,w); g(e3,transformed p,penoffset-(1,1.3)of(pencircle scaled20 yscaled-.5),
 directiontime (0,1) of ((0,0)..controls(1,1)and($,1)..(3,0)),
 point 3.14159 of p~9 intersectiontimes subpath (3.14159,4) of p~9,
 (($,1.1)..(1,$)) intersectiontimes precontrol$ of (0,0);
addto e3 doublepath(-4094.99998,0)..(4094.99998,-.00001) withpen penoffset 0 of
pencircle; addto e3 also e3 shifted (0,257); ,"flushed with pride"; numeric xx;
def f(expr x,y,z)=showdependencies;tracingcapsules:=1;showdependencies;show
 1/3(3,6)*((x+y)+(y-x)), (1,1)/sqrt2 zscaled (x+1,x+2) - (x+1,x+2) rotated 45,
 (0,1) zscaled (1,y+2)-(1,y+2) rotated 90 enddef; f((xx+1)/.3,(yy-1)/.5,(xx,0));
'=(1000o3-4000(o1-o2)+4000o2+9,-.01o3+3ooo+
 [[oo=9/10(o2+o4+o5-20);o4=o5=8/9(o1+.5o2); o6=-.0001o2;showdependencies;
  numeric o[];xpart(alfa,[[pair alfa;0]])]])=-2/3[[save p;(p$,1-p$)]];
xoffset:=yoffset:=4000[[oo=.5ooo=2*-1/2(ooo+[[numeric ooo;1]]);oo]];
for @=angle(sqrt$,mlog$):charext:=uniformdeviate$;charht:=2048;granularity:=-8;
addto e3 contour (0,-100)..tension 500..(100,-99)..tension 3000..cycle;
tracingoutput:=@; shipout e3; special "bye"; interim char 99 = "c"; true=false;
[[clear(tracingcommands); charcode:=ASCII char 269-13; shipout+nullpicture;
"careful" quote for for = @ step 200 until 2*2600: &" METAFONT" endfor;]];
```

```
158 scrollmode; "hello again"&char31; save p; fillin:=-.043;
159 def f expr x=let )=]; let [=(; show _ enddef; begingroup tracingspecs:=1;
160 show nullpen, makepath.qq, makepath(q rotated1), makepath pencircle rotated $;
161 addto e0 doublepath (0,2){0,$}..{0,$}(0,1)..{1,0}(3,0)..(4,0){1,0}..cycle
162  withpen makepen((0,0)..(5,2.9)..(4,3)..cycle); tracingonline:=1; f xx[1);
163 showdependencies; qq:=q; showstats; bye endtext
164 % things not tested:
165 % interaction (error insertion/deletion, interrupts, \pausing, files not there)
166 % date, time; initialization of random number generator without randomseed
167 % system-dependent parsing of file names, areas, extensions
168 % certain error messages, especially fatal ones
169 % things that can't happen in INIMF
170 % unusual cases of fixed-point arithmetic
```

**Appendix C: The TRAPIN.LOG file.** When INIMF makes the TRAP.BASE file, it also creates a file called TRAP.LOG that looks like this.

```
This is METAFONT, Version 2.71 (INIMF)  25 JAN 1992 08:58
**\input trap
(trap.mf
>> << == >> ::: ||`` ''--!! ??## && @@ $$[[]]{{ }}((5.5 0.5))
>> ".."
! Not implemented: (unknown numeric)++(string).
<to be read again>
                   ;
l.4 ...}} . (( 5.5.5 )) ++ "..";

I'm afraid I don't know how to apply that operation to that
combination of types. Continue, and I'll return the second
argument (see above) as the result of the operation.

..
! Missing `=' has been inserted.
<to be read again>
                   ,
l.5   begingroup save =; let=,
                               ; save,; newinternal $=,; let )...
You should have said `let symbol = something'.
But don't worry; I'll pretend that an equals sign
was present. The next token I read will be `something'.

> errorstopmode=errorstopmode
> readstring=readstring
> 2
> "2"
> ,=,
> (=tag
> )=,
<< == >> ::: ||`` ''--!! ??## && @@ $$[[]]{{ }}(([][]))=numeric
<< == >> ::: ||`` ''--!! ??## && @@ $$[[]]{{ }}((5.5 0.5))=<< == >> :::
||`` ''--!! ??## && @@ $$[[]]{{ }}((5.5 0.5))
> year=month
! OK.
l.6 ...ring,2,"2",,,(,),<<,year;


! Missing `:' has been inserted.
<to be read again>
                   ;
l.8 ...not cycle "":1.1 forever;
                                 fi;
The next thing in this loop should have been a `:'.
So I'll pretend that a colon was present;
everything from here to `endfor' will be iterated.

{fi}
{exitif}
{[repeat the loop]}
```

```
{false}
{fi}
{exitif}
{[repeat the loop]}
{(2.1)>(2)}
{true}
{tracingcommands:=2.1}
{showtoken}
> |=:|>=|=:|>
! OK.
l.9 ... endfor; showtoken |=:|>;


! Arithmetic overflow.
l.10   tracingedges:=1/.00001
                              ; tracingequations:=$+1; p~=trac...
Uh, oh. A little while ago one of the quantities that I was
computing got too large, so I'm afraid your answers will be
somewhat askew. You'll probably have to adopt different
tactics next time. But I shall try to carry on anyway.

{tracingedges:=32767.99998}
{(0)+(1)}
{tracingequations:=1}
{(32767.99998)+(0.00002)}
! Arithmetic overflow.
l.10 ... p~=tracingedges+.00001;

Uh, oh. A little while ago one of the quantities that I was
computing got too large, so I'm afraid your answers will be
somewhat askew. You'll probably have to adopt different
tactics next time. But I shall try to carry on anyway.

{(p~)=(32767.99998)}
## p~=32767.99998
{interim}
{tracingmacros:=1}
{tracingoutput:=1}
{warningcheck:=1}
{tracingstats:=1}
{tracingchoices:=1}
{tracingpens:=1}
{tracingspecs:=1}
{ASCII("")}
{$:=-1}
{$:=x}
>> x
! Internal quantity `$' must receive a known value.
<to be read again>
                   ;
l.12 ...os:=1; $:=ASCII""; $:=x;
                                  p~:=p~;
I can't set an internal quantity to anything but a known
```

```
numeric value, so I'll have to ignore this assignment.

{p~:=32767.99998}
## p~=32767.99998
! Value is too large (32767.99998).
l.12 ...:=ASCII""; $:=x; p~:=p~;

The equation I just processed has given some variable
a value of 4096 or more. Continue and I'll try to cope
with that big value; but it might be dangerous.
(Set warningcheck:=0 to suppress this message.)

{delimiters}
{delimiters}
{vardef}
{def}

foo(TEXT2)->begingroup(TEXT2)endgroup
(SUFFIX0)<-
(SUFFIX1)<-foo
(TEXT2)<-interim.proofing:=(-0.5;shipout.nullpicture;special"3"
{begingroup}
{interim}
{-(0.5)}
! Missing `)' has been inserted.
<to be read again>
                   ;
<argument> ...m.proofing:=(-0.5;
                                shipout.nullpicture;special"3"
foo->begingroup(TEXT2)
                      endgroup
l.16 ...ture; special"3" endtext
                                ;
I found no right delimiter to match a left one. So I've
put one in, behind the scenes; this may fix the problem.

{proofing:=-0.5}
{shipout}
{nullpicture}
{special}
{restoring proofing=0}
{endgroup}
{for}
{showstats}
Memory usage 291&41 (1922 still untouched)
String usage 26&83 (895&11534 still untouched)
! OK.
l.17 ...ing:fi endfor showstats;
                                 let!!=skipto;

{let}
{path}
{-(15)}
```

```
{-(12)}
Path at line 18, before choices:
(0,0)..controls (15,4) and (-15,-12)
 ..(4,0)

Path at line 18, after choices:
(0,0)..controls (15,4) and (-15,-12)
 ..(4,0)

{(unknown path p~)=(path)}
{everyjob}
{vardef}
{let}
{vardef}
{def}
{elseif}
 )
Beginning to dump on file trap.base
 (preloaded base=trap 92.1.25)
1113 strings of total length 20516
395 memory locations dumped; current usage is 317&67
265 symbolic tokens
```

**Appendix D: The TRAP.LOG file.** Here is the major output of the TRAP test; it is generated by running INIMF and loading TRAP.BASE, then reading TRAP.MF.

```
This is METAFONT, Version 2.71 (preloaded base=trap 92.1.25)  25 JAN 1992 08:58
** &trap  trap
(trap.mf
{if}
{known(0)}
{not(true)}
{false}
{known("")}
{true}

/*\‘’@#->begingroup.message(SUFFIX1)&str(SUFFIX0)&jobname&char.ASCII’‘&s
tr(SUFFIX2)!endgroup
(SUFFIX0)<-/*\
(SUFFIX1)<-‘’
(SUFFIX2)<-pass2
{begingroup}
{message}

‘’->begingroup’‘endgroup
(SUFFIX0)<-
(SUFFIX1)<-‘’
{begingroup}

’‘->"\*/"
{endgroup}
{("\*/")&("/*\")}
{jobname}
{("\*//*\")&("trap")}

’‘->"\*/"
{ASCII("\*/")}
{char(92)}
{("\*//*\trap")&("\")}
{("\*//*\trap\")&("pass2!")}

\*//*\trap\pass2!
{endgroup}
{outer}
{let}
{delimiters}

! Missing symbolic token inserted.
<inserted text>
                INACCESSIBLE
l.21 ...t next=\; delimiters ^~7
                                 ! fi
Sorry: You can’t redefine a number, string, or expr.
I’ve inserted an inaccessible symbol so that your
definition will be completed without mixing me up too badly.

! Extra tokens will be flushed.
<to be read again>
                   !
l.21 ... next=\; delimiters ^~7!
                                  fi
I’ve just read as much of that statement as I could fathom,
so a semicolon should have been next. It’s very puzzling...
but I’ll try to get myself back together, by ignoring
everything up to the next ‘;’. Please insert a semicolon
now in front of anything that you don’t want me to delete.
(See Chapter 27 of The METAFONTbook for an example.)

! Forbidden token found while scanning to the end of the statement.
<inserted text>
                ;
<to be read again>
                   \
l.22 next\
```

```
          ; % the second pass will now compute silently; the ...
A previous error seems to have propagated,
causing me to read past where you wanted me to stop.
I'll try to recover; but if the error is serious,
you'd better type 'E' or 'X' now and fix your file.

{\}
{batchmode}

! An expression can't begin with 'endgroup'.
<inserted text>
                0
<to be read again>
                   endgroup
l.23 batchmode; ^~7,endgroup
                              pausing:=1; exitif p exitif bool...
I'm afraid I need some sort of value in order to continue,
so I've tentatively inserted '0'. You may want to
delete this zero and insert something else;
see Chapter 27 of The METAFONTbook for an example.

! Missing ' INACCESSIBLE' has been inserted.
<to be read again>
                   endgroup
l.23 batchmode; ^~7,endgroup
                              pausing:=1; exitif p exitif bool...
I found no right delimiter to match a left one. So I've
put one in, behind the scenes; this may fix the problem.

! Extra 'endgroup'.
<recently read> endgroup

l.23 batchmode; ^~7,endgroup
                              pausing:=1; exitif p exitif bool...
I'm not currently working on a 'begingroup',
so I had better not try to end anything.

{pausing:=1}
{exitif}
{exitif}
{pencircle}
{endfor}
! Extra 'endfor'.
l.23 ...ean pen pencircle endfor

I'm not currently working on a for loop,
so I had better not try to end anything.

{scantokens}
{begingroup}
{message}
{char(0)}
{("^^@")&("watch this")}
^^@watch this
{-(1)}
{char(-1)}
{("pair p[],';")&("^^ff")}
{endgroup}
{pen(future pen)}
{boolean(true)}
{true}
! No loop is in progress.
<to be read again>
                   pair
<scantokens> pair
                  p[],';^^ff
<to be read again>
                    path
l.25 path
          p[][]p,w,qw; qw=(1,-2)..(2,-1)..(2.5,0.5)..(1,2)..(...
Why say 'exitif' when there's nothing to exit from?
```

```
>> p
! Undefined condition will be treated as ‘false’.
<to be read again>
                   pair
<scantokens> pair
                  p[],’;^^ff
<to be read again>
                   path
l.25 path
          p[][]p,w,qw; qw=(1,-2)..(2,-1)..(2.5,0.5)..(1,2)..(...
The expression shown above should have had a definite
true-or-false value. I’m changing it to ‘false’.

{false}
! Missing ‘;’ has been inserted.
<to be read again>
                   pair
<scantokens> pair
                  p[],’;^^ff
<to be read again>
                   path
l.25 path
          p[][]p,w,qw; qw=(1,-2)..(2,-1)..(2.5,0.5)..(1,2)..(...
After ‘exitif <boolean exp>’ I expect to see a semicolon.
I shall pretend that one was there.

{pair}
! Text line contains an invalid character.
<scantokens> pair p[],’;^^ff

<to be read again>
                   path
l.25 path
          p[][]p,w,qw; qw=(1,-2)..(2,-1)..(2.5,0.5)..(1,2)..(...
A funny symbol that I can’t read has just been input.
Continue, and I’ll forget that it ever happened.

{path}
{-(2)}
{-(1)}
{turningnumber((xpart ’,ypart ’))}
Path at line 25, before choices:
(1,-2){curl 1}
 ..(2,-1)
 ..(2.5,0.5)
 ..(1,2)
 ..{curl 1}(0,2.5)

Path at line 25, after choices:
(1,-2)..controls (1.37755,-1.71404) and (1.71404,-1.37755)
 ..(2,-1)..controls (2.33353,-0.55965) and (2.59729,-0.04124)
 ..(2.5,0.5)..controls (2.36812,1.23369) and (1.6712,1.65662)
 ..(1,2)..controls (0.66821,2.16974) and (0.33485,2.33641)
 ..(0,2.5)

{(unknown path qw)=(path)}
{numeric}
! Enormous number has been reduced.
l.26 ...[$] ]]=10000000000000000
                                ; "this string constant is in...
I can’t handle numbers bigger than about 4095.99998;
so I’ve changed your constant to that maximum amount.

{(p[[ [-1] ]])=(4095.99998)}
## p[[ [-1] ]]=4095.99998
! Incomplete string token has been flushed.
l.26 ...g constant is incomplete

Strings should finish on the same line as they began.
I’ve deleted the partial string; you might want to
```

```
insert another by typing, e.g., ‘I"new string"’.

{string}
! Declared variable conflicts with previous vardef.
<to be read again>
                   ,
l.27 string foo[]p,
                  p~if true:[];
You can’t use, e.g., ‘numeric foo[]’ after ‘vardef foo’.
Proceed, and I’ll ignore the illegal redeclaration.

{if}
{true}
{true}
{boolean}
{fi}
{showvariable}
boolean.boolean=unknown boolean
! OK.
l.28 ...n; showvariable boolean;
                                 def\\= =end enddef;

{def}
{picture}
{pen}
! Illegal suffix of declared variable will be flushed.
<to be read again>
                   [
<to be read again>
                   "a"
l.30 pen p~[]~,q["a"
                    ,qq; p~1~=q=pencircle scaled mexp(-3016.5...
Variables in declarations must consist entirely of
names and collective subscripts, e.g., ‘x[]a’.
Are you trying to use a reserved word in a variable name?
I’m going to discard the junk I found here,
up to the next comma or the end of the declaration.

{pencircle}
{-(3016.57654)}
{mexp(-3016.57654)}
{(future pen)scaled(0)}
Pen polygon at line 30 (newly created):
(0.5,0)
 .. (0,0.5)
 .. (-0.5,0)
 .. (0,-0.5)
 .. cycle

{(unknown pen q)=(pen)}
{(unknown pen p~1~)=(pen)}
{transform}
! Illegal suffix of declared variable will be flushed.
<to be read again>
                   0
l.31 transform p,pp0
                    ; if p=p:qq=makepen((1,0)..cycle) xscaled...
Variables in declarations must consist entirely of
names and collective subscripts, e.g., ‘x[]a’.
Explicit subscripts like ‘x15a’ aren’t permitted.
I’m going to discard the junk I found here,
up to the next comma or the end of the declaration.

{if}
{((xpart p,ypart p,xxpart p,xypart p,yxpart p,yypart p))=((xpart p,ypart
 p,xxpart p,xypart p,yxpart p,yypart p))}
{true}
Path at line 31, before choices:
(1,0)
 ..cycle
```

```
Path at line 31, after choices:
(1,0)..controls (1,0) and (1,0)
 ..cycle

{makepen(path)}
{hex("1000")}
! Number too large (4096).
<to be read again>
                   ;
l.31 ...cle) xscaled hex "1000";
                                fi
I have trouble with numbers greater than 4095; watch out.

{(future pen)xscaled(4096)}
! Pen too large.
<to be read again>
                   ;
l.31 ...cle) xscaled hex "1000";
                                fi
The cycle you specified has a coordinate of 4095.5 or more.
So I've replaced it by the trivial path `(0,0)..cycle'.

Pen polygon at line 31 (newly created):
(0,0)
 .. cycle

{(unknown pen qq)=(pen)}
{fi}
Path at line 32, before choices:
(0,0)
 ..(1,0)
 ..(0,1)
 ..(0,0)
 ..(1,0)
 ..(0,1)
 ..cycle

Path at line 32, after choices:
(0,0)..controls (0.29056,-0.29056) and (0.75859,-0.30772)
 ..(1,0)..controls (1.51964,0.66237) and (0.66237,1.51964)
 ..(0,1)..controls (-0.30772,0.75859) and (-0.29056,0.29056)
 ..(0,0)..controls (0.29056,-0.29056) and (0.75859,-0.30772)
 ..(1,0)..controls (1.51964,0.66237) and (0.66237,1.51964)
 ..(0,1)..controls (-0.30772,0.75859) and (-0.29056,0.29056)
 ..cycle

{makepen(path)}
! Pen cycle must be convex.
<to be read again>
                   ;
l.32 ...)..(1,0)..(0,1)..cycle);

The cycle you specified either has consecutive equal points
or turns right or turns through more than 360 degrees.
So I've replaced it by the trivial path `(0,0)..cycle'.

Pen polygon at line 32 (newly created):
(0,0)
 .. cycle

{qq:=pen}
{vardef}
! Missing parameter type; `expr' will be assumed.
<to be read again>
                   )
l.33 ...ext suffix a,b endtext()
                                )suffix@=show #@; p.a.b() end...
You should've had `expr' or `suffix' or `text' here.

{expandafter}
{\}
```

```
{let}

\\->=end
{outer}
{pencircle}
{(future pen)scaled(4.5)}
{(future pen)yscaled(2)}
Pen polygon at line 34 (newly created):
(0.5,-4.5)
 .. (1,-4)
 .. (2,-2.5)
 .. (2.5,0)
 .. (2,2.5)
 .. (1,4)
 .. (0.5,4.5)
 .. (-0.5,4.5)
 .. (-1,4)
 .. (-2,2.5)
 .. (-2.5,0)
 .. (-2,-2.5)
 .. (-1,-4)
 .. (-0.5,-4.5)
 .. cycle

{qq:=pen}
{((6,12))-((xpart p7,ypart p7))}
{((0,1))transformed((xpart p,ypart p,xxpart p,xypart p,yxpart p,yypart p
))}
{(x)-(x)}
{(2)/(0)}
>> 2
! Division by zero.
<to be read again>
                   ,
l.35 ...)transformed p=(2/(x-x),
                                3/0)transformed p;
You're trying to divide the quantity shown above the error
message by zero. I'm going to divide it by one instead.

! Division by zero.
l.35 ...ansformed p=(2/(x-x),3/0
                                )transformed p;
I'll pretend that you meant to divide by 1.

{((2,3))transformed((xpart p,ypart p,xxpart p,xypart p,yxpart p,yypart p
))}
{((linearform,linearform))=((linearform,linearform))}
## yxpart p=-yypart p
## xxpart p=-xypart p
{((-xpart p7+6,-ypart p7+12))=((linearform,linearform))}
## ypart p7=-ypart p-yypart p+12
## xpart p7=-xpart p-xypart p+6
{\}
{if}
{string(unknown string p~[-1])}
{true}
{(p0.1 0.2)-(p0.1 0.2)}
! The token `endtext' is no longer a right delimiter.
l.36 ...1.2-p.1.199999,1 endtext
                                transformed p;
Strange: This token has lost its former meaning!
I'll read it as a right delimiter this time;
but watch out, I'll probably miss it later.

{((0,1))transformed((xpart p,ypart p,-xypart p,xypart p,-yypart p,yypart
 p))}
{((linearform,linearform))=((linearform,linearform))}
## ypart p=-yypart p+6
#### ypart p7=6
## xpart p=-xypart p+3
#### xpart p7=3
```

```
{(unknown path p1 2p)=((3,6))}
{showstopping:=0}
{showvariable}
p=(-xypart p+3,-yypart p+6,-xypart p,xypart p,-yypart p,yypart p)
p[]=pair
p[][]=numeric
p[][]p=unknown path
p[][]p~=macro:(SUFFIX2)(SUFFIX3)(EXPR4)<suffix>->begingroup ETC.
p[]~=unknown boolean
p~=path
p~[]=unknown string
p~[]~=unknown pen
p~[]~[][]=unknown picture
p~[-1]=unknown string p~[-1]
p~1~=pen
p[[ [] ]]=numeric
p[[ [-1] ]]=4095.99998
p0.1 0.2=p0.1 0.2
p1 2p=path
p7=(3,6)
{((-xypart p+3,-yypart p+6,-xypart p,xypart p,-yypart p,yypart p))=((-xy
part p+3,-yypart p+6,-xypart p,xypart p,-yypart p,yypart p))}
{let}
{let}
{xxpart((-xypart p+3,-yypart p+6,-xypart p,xypart p,-yypart p,yypart p))
}
{(-xypart p)+(0.002)}
{yxpart((-xypart p+3,-yypart p+6,-xypart p,xypart p,-yypart p,yypart p))
}
{((xpart p2,ypart p2))-((xpart p1,ypart p1))}
{(1)*((linearform,linearform))}
{((xpart p1,ypart p1))+((linearform,linearform))}
{(y)+(0.00002)}
{yypart((-xypart p+3,-yypart p+6,-xypart p,xypart p,-yypart p,yypart p))
}
{xypart((-xypart p+3,-yypart p+6,-xypart p,xypart p,-yypart p,yypart p))
}
{((5,y))=((yypart p,xypart p))}
## y=xypart p
## yypart p=5
#### ypart p=1
#### yxpart p=-5
{((5,xypart p+0.00002))=((5,xypart p))}
{((xpart p2,ypart p2))=((5,xypart p))}
## ypart p2=xypart p
## xpart p2=5
{((-xypart p+0.002,-5))=((5,xypart p))}
## xypart p=-5
#### ypart p2=-5
#### y=-5
#### xpart p=8
#### xxpart p=5
! Inconsistent equation (off by -0.002).
<to be read again>
                   ;
l.38 ...,y)=(yypart p,xypart p);

The equation I just read contradicts what was said before.
But don't worry; continue and I'll just ignore it.

Path at line 39, before choices:
(0,0)..controls (15,4) and (-15,-12)
 ..(4,0)
 ..cycle

Path at line 39, after choices:
(0,0)..controls (15,4) and (-15,-12)
 ..(4,0)..controls (17.52783,8.54388) and (-15.45978,-4.12262)
 ..cycle

{reverse(path)}
```

```
{(path)transformed((8,1,5,-5,-5,5))}
{(path)=(unknown path p2 3p)}

p1 2p~(SUFFIX2)(SUFFIX3)(EXPR4)<suffix>->begingroup.show(SUFFIX0);p(SUFF
IX2)(SUFFIX3)((EXPR4)endgroup
(SUFFIX0)<-p1 2p
(SUFFIX1)<-~
(SUFFIX2)<-
(SUFFIX3)<-2 3p~
! A primary expression can't begin with `right delimiter that matches ('
.
<inserted text>
                0
<to be read again>
                   )
l.40 ...000000001]2p~(,[2]3p~,-)
                                  =p~1~2[pausing];
I'm afraid I need some sort of value in order to continue,
so I've tentatively inserted `0'. You may want to
delete this zero and insert something else;
see Chapter 27 of The METAFONTbook for an example.

{-(0)}
(EXPR4)<-0
(SUFFIX5)<-
{begingroup}
{show}
>> Path at line 40:
(3,6)


p2 3p~(SUFFIX2)(SUFFIX3)(EXPR4)<suffix>->begingroup.show(SUFFIX0);p(SUFF
IX2)(SUFFIX3)((EXPR4)endgroup
(SUFFIX0)<-p2 3p
(SUFFIX1)<-~
! Missing `,' has been inserted.
<to be read again>
                   (0)
p1 2p~->...IX2)(SUFFIX3)((EXPR4)
                                  endgroup
<to be read again>
                   =
l.40 ...00000001]2p~(,[2]3p~,-)=
                                  p~1~2[pausing];
I've finished reading a macro argument and am about to
read another; the arguments weren't delimited correctly.
You might want to delete some tokens before continuing.

(SUFFIX2)<-
! Missing `,' has been inserted.
<to be read again>
                   (0)
p1 2p~->...IX2)(SUFFIX3)((EXPR4)
                                  endgroup
<to be read again>
                   =
l.40 ...00000001]2p~(,[2]3p~,-)=
                                  p~1~2[pausing];
I've finished reading a macro argument and am about to
read another; the arguments weren't delimited correctly.
You might want to delete some tokens before continuing.

(SUFFIX3)<-
! Missing `)' has been inserted.
<to be read again>
                   endgroup
<to be read again>
                   =
l.40 ...00000001]2p~(,[2]3p~,-)=
                                  p~1~2[pausing];
I've gotten to the end of the macro parameter list.
```

```
You might want to delete some tokens before continuing.

(EXPR4)<-0
(SUFFIX5)<-
{begingroup}
{show}
>> Path at line 40:
(8,1)..controls (-48.68579,57.68579) and (52.91974,-43.91974)
 ..(28,-19)..controls (-7,16) and (63,-54)
 ..cycle

>> (8,1,5,-5,-5,5)
! Isolated expression.
<to be read again>
                   (
p2 3p~->...;p(SUFFIX2)(SUFFIX3)(
                                  (EXPR4)endgroup
<to be read again>
                   endgroup
<to be read again>
                   =
l.40 ...00000001]2p~(,[2]3p~,-)=
                                  p~1~2[pausing];
I couldn't find an `=' or `:=' after the
expression that is shown above this error message,
so I guess I'll just ignore it and carry on.

! Extra tokens will be flushed.
<to be read again>
                   (
p2 3p~->...;p(SUFFIX2)(SUFFIX3)(
                                  (EXPR4)endgroup
<to be read again>
                   endgroup
<to be read again>
                   =
l.40 ...00000001]2p~(,[2]3p~,-)=
                                  p~1~2[pausing];
I've just read as much of that statement as I could fathom,
so a semicolon should have been next. It's very puzzling...
but I'll try to get myself back together, by ignoring
everything up to the next `;'. Please insert a semicolon
now in front of anything that you don't want me to delete.
(See Chapter 27 of The METAFONTbook for an example.)

{endgroup}
{endgroup}
{(vacuous)=(unknown picture p~1~2 1)}
>> vacuous
>> unknown picture p~1~2 1
! Equation cannot be performed (vacuous=unknown picture).
<to be read again>
                   ;
l.40 ...2]3p~,-)=p~1~2[pausing];

I'm sorry, but I don't know how to make such things equal.
(See the two expressions just above the error message.)

{vardef}
Runaway definition?
if.p(SUFFIX1)(SUFFIX1)=(SUFFIX1)(SUFFIX1)p.fi
! Forbidden token found while scanning the definition of p~[].
<inserted text>
                enddef
<to be read again>
                   ;
l.41 ...iary t:=if p@ @=@ @p fi;
                                   vardef p[][]p~[]=BAD; inner ;;
I suspect you have forgotten an `enddef',
causing me to read past where you wanted me to stop.
I'll try to recover; but if the error is serious,
```

```
you'd better type 'E' or 'X' now and fix your file.

{vardef}
! This variable already starts with a macro.
l.41 ...@p fi; vardef p[][]p~[]=
                                BAD; inner ;;
After 'vardef a' you can't say 'vardef a.b'.
So I'll have to discard this definition.

Runaway definition?
BAD
! Forbidden token found while scanning the definition of a bad variable.
<inserted text>
                enddef
<to be read again>
                   ;
l.41 ...i; vardef p[][]p~[]=BAD;
                                 inner ;;
I suspect you have forgotten an 'enddef',
causing me to read past where you wanted me to stop.
I'll try to recover; but if the error is serious,
you'd better type 'E' or 'X' now and fix your file.

{inner}
{show}
{-(2)}

p~[-2]@#<tertiary>->begingroup.if.p(SUFFIX1)(SUFFIX1)=(SUFFIX1)(SUFFIX1)
p.fi.endgroup
(SUFFIX0)<-p~
(SUFFIX1)<-[-2]
(SUFFIX2)<-~
! A tertiary expression can't begin with '['.
<inserted text>
                0
<to be read again>
                   [
<to be read again>
                   (3000)
<to be read again>
                   ,
l.42 show p~[-2]~[3000,
                      x]++4000>path p3; showvariable p,P;
I'm afraid I need some sort of value in order to continue,
so I've tentatively inserted '0'. You may want to
delete this zero and insert something else;
see Chapter 27 of The METAFONTbook for an example.

{(x)-(3000)}
{(0)*(x-3000)}
{(3000)+(0)}
{(3000)++(4000)}
(EXPR3)<-4999.99998
{begingroup}
{if}
{(p[-2][-2])=(-2)}
>> p[-2][-2]+2
! Unknown relation will be considered false.
<to be read again>
                   [-2]
p~[-2]->...1)=(SUFFIX1)(SUFFIX1)
                                p.fi.endgroup
<to be read again>
                   >
l.42 show p~[-2]~[3000,x]++4000>
                                path p3; showvariable p,P;
Oh dear. I can't decide if the expression above is positive,
negative, or zero. So this comparison test won't be 'true'.

{false}
! Missing ':' has been inserted.
```

```
<to be read again>
                   [-2]
p~[-2]->...1)=(SUFFIX1)(SUFFIX1)
                                  p.fi.endgroup
<to be read again>
                   >
l.42 show p~[-2]~[3000,x]++4000>
                                  path p3; showvariable p,P;
There should've been a colon after the condition.
I shall pretend that one was there.

{endgroup}
{path((xpart p3,ypart p3))}
{(vacuous)>(false)}
>> vacuous
>> false
! Not implemented: (vacuous)>(boolean).
<to be read again>
                   ;
l.42 ...~[3000,x]++4000>path p3;
                                   showvariable p,P;
I'm afraid I don't know how to apply that operation to that
combination of types. Continue, and I'll return the second
argument (see above) as the result of the operation.

>> false
{showvariable}
p=(8,1,5,-5,-5,5)
p[]=pair
p[][]=numeric
p[][]p=unknown path
p[][]p~=macro:(SUFFIX2)(SUFFIX3)(EXPR4)<suffix>->begingroup ETC.
p[]~=unknown boolean
p~=path
p~[]@#=macro:<tertiary>->begingroup.if.p(SUFFIX1)(SUFFIX1) ETC.
p[[ [] ]]=numeric
p[[ [-1] ]]=4095.99998
p[-2][-2]=p[-2][-2]
p0.1 0.2=p0.1 0.2
p1=(xpart p1,ypart p1)
p1 2p=path
p2=(5,-5)
p2 3p=path
p3=(xpart p3,ypart p3)
p7=(3,6)
> P=tag
{numeric}
{(2)*(alpha)}
{(p3~)=(2alpha)}
## alpha=0.5p3~
{(1)/(-1)}
{(3)*(beta)}
{(p[-1]~)=(3beta)}
## beta=0.33333p[-1]~
{begingroup}
{save}
{showvariable}
> p=tag
{(3)*(0.33333(SAVED)p[-1]~)}
{((SAVED)p[-1]~)=(1)}
## (SAVED)p[-1]~=1
#### beta=0.33333
{restoring p}
{endgroup}
{showvariable}
p=(8,1,5,-5,-5,5)
p[]=pair
p[][]=numeric
p[][]p=unknown path
p[][]p~=macro:(SUFFIX2)(SUFFIX3)(EXPR4)<suffix>->begingroup ETC.
p[]~=numeric
```

```
p~=path
p~[]@#=macro:<tertiary>->begingroup.if.p(SUFFIX1)(SUFFIX1) ETC.
p[[ [] ]]=numeric
p[[ [-1] ]]=4095.99998
p[-2][-2]=p[-2][-2]
p[-1]~=1
p0.1 0.2=p0.1 0.2
p1=(xpart p1,ypart p1)
p1 2p=path
p2=(5,-5)
p2 3p=path
p3=(xpart p3,ypart p3)
p3~=p3~
p7=(3,6)
{def}
{def}
! Missing `=' has been inserted.
<to be read again>
                   false
l.45 ...enddef;def!primary!false
                                  ):!fi enddef;
The next thing in this `def' should have been `=',
because I've already looked at the definition heading.
But don't worry; I'll pretend that an equals sign
was present. Everything from here to `enddef'
will be the replacement text of this macro.

{def}
{(path)scaled(-1)}
Path at line 46, before choices:
(1,-2)..controls (1.37755,-1.71404) and (1.71404,-1.37755)
 ..(2,-1)..controls (2.33353,-0.55965) and (2.59729,-0.04124)
 ..(2.5,0.5)..controls (2.36812,1.23369) and (1.6712,1.65662)
 ..(1,2)..controls (0.66821,2.16974) and (0.33485,2.33641)
 ..(0,2.5)
 ..(-1,2)..controls (-1.37755,1.71404) and (-1.71404,1.37755)
 ..(-2,1)..controls (-2.33353,0.55965) and (-2.59729,0.04124)
 ..(-2.5,-0.5)..controls (-2.36812,-1.23369) and (-1.6712,-1.65662)
 ..(-1,-2)..controls (-0.66821,-2.16974) and (-0.33485,-2.33641)
 ..(0,-2.5)
 ..cycle

Path at line 46, after choices:
(1,-2)..controls (1.37755,-1.71404) and (1.71404,-1.37755)
 ..(2,-1)..controls (2.33353,-0.55965) and (2.59729,-0.04124)
 ..(2.5,0.5)..controls (2.36812,1.23369) and (1.6712,1.65662)
 ..(1,2)..controls (0.66821,2.16974) and (0.33485,2.33641)
 ..(0,2.5)..controls (-0.37186,2.68167) and (-0.668,2.25146)
 ..(-1,2)..controls (-1.37755,1.71404) and (-1.71404,1.37755)
 ..(-2,1)..controls (-2.33353,0.55965) and (-2.59729,0.04124)
 ..(-2.5,-0.5)..controls (-2.36812,-1.23369) and (-1.6712,-1.65662)
 ..(-1,-2)..controls (-0.66821,-2.16974) and (-0.33485,-2.33641)
 ..(0,-2.5)..controls (0.37186,-2.68167) and (0.668,-2.25146)
 ..cycle

{makepen(path)}
Pen polygon at line 46 (newly created):
(1,-2)
 .. (2,-1)
 .. (2.5,0.5)
 .. (1,2)
 .. (0,2.5)
 .. (-1,2)
 .. (-2,1)
 .. (-2.5,-0.5)
 .. (-1,-2)
 .. (0,-2.5)
 .. cycle

{qq:=pen}
{primarydef}
```

```
{secondarydef}

//<expr>->

//<expr>->
{pencircle}
{length(path)}
{(future pen)slanted(1)}
{((3,6))-((5,-5))}

_aa__<secondary>->if(true
{(0.1)*(15)}
{odd(1.50009)}
{not(false)}
{known((8,1,5,-5,-5,5))}
{(true)and(true)}
(EXPR0)<-true
{if}
{true}

!<primary>->false):(EXPR0)fi
(EXPR0)<-(5,-5)
{false}
{(true)or(false)}
{true}
{fi}
{-((5,-5))}
{-((5,-5))}
{+(1)}
{-(1)}
>> -1
! Improper curl has been replaced by 1.
<to be read again>
                   )
l.51  {curl- +1)
                ..tension atleast1..cycle sqrt2++sqrt2***[[]];
A curl must be a known, nonnegative number.

! Missing `}' has been inserted.
<to be read again>
                   )
l.51  {curl- +1)
                ..tension atleast1..cycle sqrt2++sqrt2***[[]];
I've scanned a direction spec for part of a path,
so a right brace should have come next.
I shall pretend that one was there.

Path at line 51, before choices:
(-5,5)

Path at line 51, after choices:
(-5,5)

Path at line 51, before choices:
(3,6)..controls (5,-5) and (-5,5)
 ..(-5,5)..tension atleast1
 ..{2896.30943,-2896.30934}cycle

Path at line 51, after choices:
(3,6)..controls (5,-5) and (-5,5)
 ..(-5,5)..controls (-3.29726,7.86205) and (0.64516,8.35484)
 ..cycle

! Missing `)' has been inserted.
<to be read again>
                   sqrt
l.51 ...ion atleast1..cycle sqrt
                                2++sqrt2***[[]];
I found no right delimiter to match a left one. So I've
put one in, behind the scenes; this may fix the problem.
```

```
{((-2,11))subpath(path)}
{reverse(path)}
{makepen(path)}

**->[[show(EXPR0)*(EXPR1)]]
(EXPR0)<-future pen
(EXPR1)<-future pen
{begingroup}
{show}
{(future pen)*(future pen)}
>> future pen
>> future pen
! Not implemented: (future pen)*(future pen).
<to be read again>
                   ]]
<to be read again>
                   sqrt
l.51 ...ion atleast1..cycle sqrt
                                2++sqrt2***[[]];
I'm afraid I don't know how to apply that operation to that
combination of types. Continue, and I'll return the second
argument (see above) as the result of the operation.

! Pen path must be a cycle.
<to be read again>
                   ]]
<to be read again>
                   sqrt
l.51 ...ion atleast1..cycle sqrt
                                2++sqrt2***[[]];
I can't make a pen from the given path.
So I've replaced it by the trivial path `(0,0)..cycle'.

>> Pen polygon at line 51:
(0,0)
 .. cycle

{endgroup}
(EXPR0)<-vacuous
{sqrt(2)}
{sqrt(2)}
{(1.41422)++(1.41422)}
{begingroup}
{endgroup}

***->expandafter(EXPR1)scantokens"**oct"(EXPR0)
(EXPR0)<-2
(EXPR1)<-vacuous
{expandafter}
{scantokens}
{oct(2)}
>> 2
! Not implemented: oct(known numeric).
<to be read again>
                   ;
l.51 ...cle sqrt2++sqrt2***[[]];

I'm afraid I don't know how to apply that operation to that
particular type. Continue, and I'll simply return the
argument (shown above) as the result of the operation.


**->[[show(EXPR0)*(EXPR1)]]
(EXPR0)<-vacuous
(EXPR1)<-2
{begingroup}
{show}
{(vacuous)*(2)}
>> vacuous
>> 2
! Not implemented: (vacuous)*(known numeric).
```

```
<to be read again>
                   ]]
<to be read again>
                   ;
l.51 ...cle sqrt2++sqrt2***[[]];

I'm afraid I don't know how to apply that operation to that
combination of types. Continue, and I'll return the second
argument (see above) as the result of the operation.

>> 2
{endgroup}
(EXPR0)<-vacuous
{begingroup}
{interim}
{-(20.5)}
{charcode:=-20.5}
{proofing:=-20.5}
{-(2048)}
{chardp:=-2048}
{shipout}
{nullpicture}
! Enormous chardp has been reduced.
<to be read again>
                   ]]
l.52 ...48;shipout nullpicture]]
                                 ;
Font metric dimensions must be less than 2048pt.

{restoring proofing=0}
{endgroup}
{if}
{-(275.50002)}
{charexists(-275.50002)}
{known(unknown path p0 0p)}
{(true)>(false)}
{known(path)}
{(true)=(true)}
{true}
{randomseed}
! Missing ':=' has been inserted.
<to be read again>
                   charcode
l.53 ... p~: randomseed charcode
                                 ; fi
Always say 'randomseed:=<numeric expression>'.

{randomseed:=-20.5}
{fi}
{randomseed}
>> "goof"
! Unknown value will be ignored.
<to be read again>
                   ;
l.54 randomseed:="goof";
                         a[($,18++1+-+18),(2,3)]=b[(3,2),(1,$);
Your expression was too random for me to handle,
so I won't change the random seed just now.

{(18)++(1)}
{(18.02776)+-+(18)}
{((2,3))-((-1,1))}
{(a)*((3,2))}
{((-1,1))+((3a,2a))}
! Missing ']' has been inserted.
<to be read again>
                   ;
l.54 ...8),(2,3)]=b[(3,2),(1,$);

I've scanned an expression of the form 'a[b,c',
so a right bracket should have come next.
```