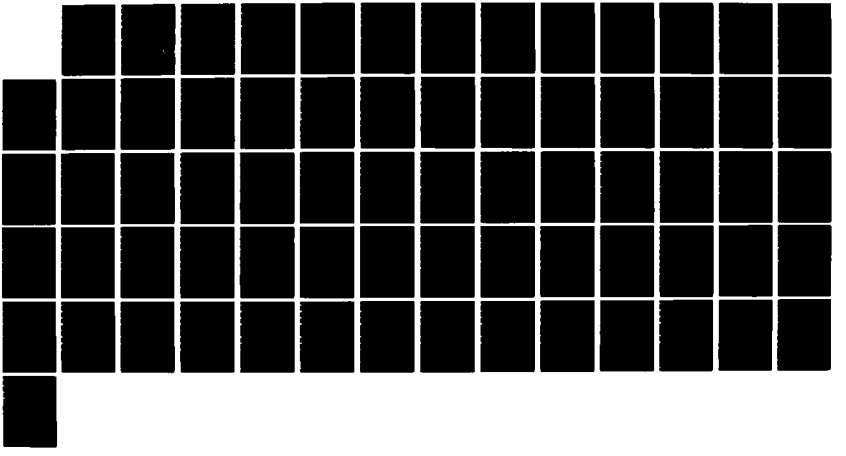
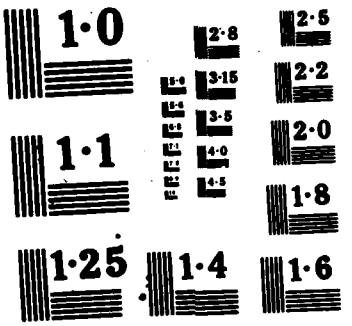
AD-A167 763 PC11 PILOT REFERENCE MANUAL(U) DEFENSE TECHNICAL 1/1
INFORMATION CENTER ALEXANDRIA VA OFFICE OF INFORMATION
SYSTEMS AND TECHNOLOGY R G THORNETT MAY 86
UNCLASSIFIED DTIC/TR-86/12 F/G 9/2 NL





PC11 PILOT REFERENCE MANUAL

Richard G. Thornett

Office of Information Systems and Technology

Cameron Station, Alexandria, VA 22304-6145

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED/UNLIMITED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) DTIC/TR-86/12	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Defense Technical Information Center 6b. OFFICE SYMBOL (If applicable) DTIC-EA	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Cameron Station Alexandria, VA 22304-6145	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO. DS000060
11. TITLE (Include Security Classification) PC11 PILOT Reference Manual	
12. PERSONAL AUTHOR(S) Thornett, Richard Geoffrey	
13a. TYPE OF REPORT 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) 86 May 15. PAGE COUNT 68
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
05	02 & 09		Computer Aided Instruction; Computer-Assisted Instruction; Training; Individualized Training; DROLS
09	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This document describes the PC11 PILOT Computer-Assisted Instruction (CAI) programming language and the PC11 PILOT interpreter which presents instruction written in that language. Two implementations of this interpreter exist: one for IBM PCs and some microcomputers compatible with them, and one for Sperry Univac 1100 Series computers. Both are written in Pascal. A course about retrieval from the Defense RDT&E Online System (DROLS) was written in PC11 PILOT. The course is available on diskettes and to dial-up users of the DTIC ADPE Time Sharing Service (DTSS). The manual also describes learner-entered commands which give help and control over the order of presentation.

(Keywords: PILOT (Programmed Instruction Learning Or Teaching); Individualized Training).

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED/UNLIMITED
22a. NAME OF RESPONSIBLE INDIVIDUAL Richard G. Thornett	22b. TELEPHONE (Include Area Code) (202) 274-7661 22c. OFFICE SYMBOL DTIC-EA

PC11 PILOT REFERENCE MANUAL

Richard G. Thornett

OFFICE OF INFORMATION SYSTEMS AND TECHNOLOGY
DEFENSE TECHNICAL INFORMATION CENTER
CAMERON STATION
ALEXANDRIA, VA 22304-6145

1. PC11 PILOT INTRODUCTION

This document describes the PC11 PILOT computer-assisted instruction (CAI) programming language and the PC11 PILOT interpreter which presents instruction written in that language. This interpreter was developed at the Defense Technical Information Center (DTIC) for a CAI project. The objective of the project is to provide both a capability for preparing and presenting CAI and one or more prototype courses. The subject of the courses is retrieval from the Defense RDT&E On-Line System (DROLS). RDT&E means Research, Development, Test and Evaluation.

1.1 PC11 PILOT

PILOT is a simple but popular CAI authoring language used since the early 1970s. PILOT is an acronym for Programmed Instruction, Learning Or Teaching. PILOT has been implemented on a wide variety of computers both large and small. There are many versions of PILOT, and many extensions and enhancements of the language are found in them.

PC11 Pilot was modeled on LHC 8080 PILOT, which is an extended version of CORE PILOT '73 and executes on microcomputers. LHC 8080 PILOT was developed by the Lister Hill National Center for Biomedical Communications. PC11 PILOT was developed by selecting a subset of the features of LHC 8080 PILOT and implementing them in Pascal 1100, using the LHC 8080 PILOT users manual as a specification. There are some differences between corresponding features of the two PILOTs.

In order to make PC11 PILOT courses widely available, two implementations were done. One is in the DTIC ADPE Time Sharing Service (DTSS), which is hosted by a Sperry Univac 1100/61 computer and is accessible through TYMNET. The other is for IBM Personal Computers (PCs) and some compatibles. The time-sharing and personal computer implementations are called PC11T and PC11P, respectively.

The PC11 PILOT Interpreter is written in the general purpose programming language Pascal, for the following reasons:

1. Pascal is an easy-to-use high-level language.
2. Pascal programs are relatively portable.
3. The C programming language is not available in the DTSS.

The PC11T PILOT interpreter is written in Pascal 1100, which is used in Sperry Univac 1100 Series computers. This Pascal was developed by M. S. Ball of the U. S. Naval Ocean Systems Center and enhanced by Ira L. Ruben of Sperry Univac. The PC11P interpreter is written in IBM Pascal.

1.2 Document Description

Section 2 of this document gives a brief overview of the processes of preparing and presenting PC11 PILOT courses for each of the two implementations. Section 3 gives an overview of the PC11 PILOT language.

Section 4 describes each PC11 PILOT command. Appendix A tells how to sign onto the DTSS and take a PC11 PILOT course. Appendix B tells how to take a PC11 PILOT course using an IBM PC. Appendix C lists error messages which might appear while preparing or presenting PC11 PILOT courses. Appendix D tells how use the MSKERMIT program to transfer files between the DTSS and an IBM PC. Appendix E is about maintenance of the PC11T PILOT course preparation and presentation system in the DTSS. Appendix F is about maintenance of the PC11P PILOT course preparation and presentation system in an IBM PC.

2. PREPARING AND PRESENTING PC11 PILOT PROGRAMS

This section briefly describes PC11 PILOT program preparation and presentation for each of the two implementations.

2.1 Preparing and Presenting PC11T PILOT Programs

PC11T PILOT lives in the DTIC ADPE Time Sharing Service (DTSS), which is hosted by a Sperry Univac 1100/61 computer. PC11T executes under the Sperry Univac 1100 Series Executive System (EXEC). The file management, text editing and other features of the EXEC were used to prepare PILOT programs. These features of the EXEC are described in the relevant Univac documents.

This section briefly describes some EXEC Control Language (ECL) commands useful in preparing and presenting PC11T PILOT instructional units. For a fuller description see the handbook "DTIC ADPE Time Sharing Service Users Guide", DLAH 4185.7.

Section 2.1.1 gives basic information on the EXEC and ECL commands. Section 2.1.2 describes ECL commands pertinent to PC11T PILOT.

2.1.1 Executive Control Language Fundamantals

2.1.1.1 Demand Run Initiation

Demand processing is a mode of operation in which processing is dependent on manual interface between the executive (EXEC) and the user during processing. It is a conversational way of operating involving repeated demand and response. Conversational operation by a remote terminal causes the EXEC or a demand processor or an active program to react and respond immediately. The terminal usually has a keyboard and a printer or cathode ray tube (CRT). A demand terminal user must turn the terminal on, set its various switches and establish the proper communication line connection if operation is on a switched line network.

Once you are connected to the DTSS, you sign onto it. This is explained in Appendix A. You start signing on by presenting a six-character site identifier to the operating system. The operating system responds and requests a user identifier and password. The character > is used to prompt you for input. When these have been entered and accepted, an RUN ECL command must be entered. An @ character is entered before each ECL command to distinguish it from data. In the DTSS the @RUN command is ordinarily entered automatically without the user seeing it. Among the parameters of the @RUN command are the user's project identifier and account number. The project identifier is the default qualifier (prefix) of the name of each file referenced in the run. Thus if your project identifier is GRAPHICS and you refer to a file as POSTERS, the full file name is GRAPHICS*POSTERS, where the asterisk separates the qualifier from the rest of the file name.

A run is a group of tasks treated as a unit by EXEC. A runstream consists of an @RUN command followed by other ECL statements and data. These control statements direct the performance of individual tasks. The tasks are executed in the order specified by the runstream. A run is terminated by the first @FIN command entered from the terminal. After @FIN enter

@@TERM to tell EXEC to disconnect you.

2.1.1.2 Naming and Creating Files

Here we consider only Univac 1100 Series disk files and their use as PILOT program files. In this context a file name is a string of one to twelve characters from the set containing A-Z, 0-9, - and $. Since $ is used in system file names, it is better not to use it in user file names. In some ECL commands a period is required after a file name, in others it is optional, and rarely it is prohibited.

To create a new file, the @ASG command is used. For example

```
@ASG,UP COURSE-2.,F40///1000
```

creates file COURSE-2 as a disk file. The @ character tells EXEC that what follows is an ECL command. ASG means assign. The U and P after the first comma indicate options. The U indicates that the file is to be cataloged unconditionally. The catalog is the EXEC's master file directory. The P indicates that the file is to be public. The F40 parameter indicates the type of mass storage equipment to be used (a type of disk drive). The space separates the command name and options from the command's parameters. Commas separate parameters and slashes separate subparameters. The number 1000 indicates that the file is to be limited to 1000 tracks. A track holds 7168 characters coded in the ASCII code. ASCII stands for the American Standard Code for Information Interchange.

A PC11 PILOT program is an ASCII text file containing a sequence of lines, each containing 80 characters or less. Lines may be entered into a file by means of the text editor (ED). To start a new text file enter, for example:

```
@ED,IQ COURSE-2.
```

Here @ED tells EXEC to start processor ED. The option letter I indicates that this is the initial insertion into the file, and Q indicates that the text is to consist of ASCII characters as opposed to Univac's Fieldata code. (The ASCII code has 128 characters, including upper and lower case letters. Fieldata has 64 characters and only upper case.) PC11 PILOT program files must be ASCII text files. File COURSE-2 must exist before it can be edited. If the initial insertion process is completed, the file's previous content (if any) will be lost.

After doing initialization, ED issues the prompt. Due to the I option, ED expects lines of text to be input. Finish entering each line by entering a return character. If you enter a nonempty line, ED inserts it into the file and again issues the prompt. An empty line is zero or more spaces followed by a return. The line entry mode of editor operation is called INPUT MODE. To change from input mode to edit mode, the enter an empty line.

In EDIT MODE a variety of commands are available for such things as moving back and forth in the file, printing lines, locating character strings, changing strings, and deleting and inserting lines. In addition to the standard editor commands, the user may form and use macro commands.

Macro commands are editor commands formed and named by the user. A macro command is a sequence of standard editor commands or macro commands or both. Macros extend and specialize the capabilities of the editor.

To end an editing session and save its result, the user enters EXIT while in edit mode. To cancel the editing session, the user enters OMIT while in edit mode. To make sure that edits are not lost, free the file edited, like this:

```
@FREE COURSE-2.
```

This command tells EXEC to update the directory entry of file COURSE-2. If this is not done before disconnection, the edits may be lost. The @FIN command frees all files assigned to the run. The FREE command is the opposite of the ASG command. ASG associates a file with a run. FREE removes such an association.

To view file COURSE-2 without changing it, enter:

```
@ED,R  COURSE-2.
```

where the R option letter specifies read only mode in which insertions and other changes are not allowed. Use this mode as a precaution against accidentally changing what you view.

PROGRAM FILES are files which contain separately accessible parts which may be referred to by name. They are called program files because many computer programs and their components can be kept in one program file. The named parts of a program file are called elements. There are three kinds of elements. SYMBOLIC elements contain text. RELOCATABLE elements contain elements produced by compilers or assemblers. ABSOLUTE ELEMENTS are executable programs or processors. An element specifier consists of a file name followed by a period followed by an element name. The rules for element names are the same as those for file names.

A symbolic program file element can be edited or printed separately.

A PC11 PILOT program is a text file which contains a sequence of PC11 PILOT commands. It is NOT a program file in the Univac 1100 Series meaning of the term.

2.1.2 PC11T PILOT Course Preparation and Presentation

A PC11 PILOT COURSE is PC11 PILOT program which is built of lessons which are built of sections which are built of units. This special lesson-section-unit (LSU) structure is explained in Section 5.1 below. For simplicity we will talk as though all PC11 PILOT programs are courses, although this is not necessarily so.

A PC11 PILOT course exists in two forms. In its first form, it is a text file which consists of a sequence of PC11 PILOT commands. In this form it can be examined and modified by the text editor ED. Its second form is a special random-access form which permits PILOT to get any course line quickly. We call the first form the EDITABLE form of the course and the

second form the EXECUTABLE form. Similarly the help file used with a course has an editable and an executable form. There is a preprocessor which converts editable PC11 PILOT courses to executable form, and another preprocessor which converts editable PC11 PILOT help files to executable form.

Preparing a PC11 PILOT course includes preparing an editable course file and help file, and inputting each of these to its preprocessor, which outputs the executable file.

Presenting a PC11 PILOT course includes specifying which executable course and help files are to be used and starting the PC11 PILOT interpreter, which reads the files, presents the course, provides help, and otherwise interacts with the learner.

2.1.2.1 PC11T PILOT Course Preparation

A PC11T PILOT course in editable form is a standard text file containing a sequence of commands written in the PC11T PILOT language. An example is the course "Introduction to DROLS Retrieval (IDR)." A course can be prepared on a word processor or microcomputer and communicated to the DTSS. The Univac text editor (ED) may be used to prepare or modify a course.

The preprocessor SEQRAN converts an editable course file into an executable course file. SEQRAN inputs file SEQFILE and outputs file RANFILE. The following ECL is used:

```
@ASG,A SEQFILE.
@ASG,A RANFILE.
@PILOT.SEQRAN
@FREE RANFILE.
```

Both files must exist. They must be assigned to the run, which statements 1 and 2 do. Statement 3 starts execution of processor SEQRAN. Statement 4 assures that the output file is saved.

A PC11T help file in editable form is a standard text file containing terms and their meanings. Each term can have up to 60 characters, including spaces. An example is ACCESSION NUMBER. A start-of-header (SOH) character (ASCII code 1) must immediately follow the last character of each term. The term's meaning begins on the next line. A blank line must follow the meaning. The terms must be in alphabetical order. An example is the help file for the course IDR. A help file may be prepared on a word processor or a microcomputer and communicated to the DTSS. The Univac text editor (ED) may be used to modify it.

The preprocessor HSEQRAN converts an editable help file into an executable help file. HSEQRAN inputs file HSEQFILE and outputs file HRANFILE. The following ECL is used:

```
@ASG,A HSEQFILE.
@ASG,A HRANFILE.
@PILOT.HSEQRAN
@FREE HRANFILE.
```

Both files must exist. They must be assigned to the run, which statements 1 and 2 do. Statement 3 starts execution of processor HSEQRAN. Statement 4 assures that the output file is saved.

The @USE command is used to tell SEQRAN or HSEQRAN what files to operate on. For example, if a course part is called PP1E (PILOT Program 1 in editable form), enter:

```
@USE SEQFILE,PP1E
@USE RANFILE,PP1
@ASG,A SEQFILE.
@ASG,A RANFILE.
@PILOT.SEQRAN
@FREE RANFILE.
```

This ECL sequence converts editable course file PP1E to executable file PP1 ready for input to PC11T PILOT. A similar sequence can convert help file PH1E to PH1.

2.1.2.2 PC11T PILOT Course Presentation

When the PC11T Pilot interpreter is started, it reads executable files PILOPROG and PILOHELP, and writes to file USERRESPONSE. The first two files were called RANFILE and HRANFILE, respectively, by the preprocessors which produced them. Each response entered by the user during the session is written to USERRESPONSE, which may be viewed later to see some of what the course and the learner did. Current practice is to use files named URFAA00, URFAA01, URFAA01, and so on through URFZZ99. To present course PP1 with help file PH1 and response file UR01 enter:

```
@USE PILOPROG,PP1
@USE PILOHELP,PH1
@USE USERRESPONSE,UR01
@ASG,A PILOPROG.
@ASG,A PILOHELP.
@ASG,A USERRESPONSE.
@PILOT.DATA
```

The first three commands give the course, help and response files the names PC11T PILOT expects. The next three assign the files to the run. The last command starts the PILOT interpreter.

Presentation of PILOPROG ends in one of the following ways:

1. PC11T PILOT encounters an END command in the course when no subroutine is active (see section 4.3 below).

2. The learner enters the user command STOP. (See section 5.2.5 below.)

3. PC11T PILOT detects an error in the course.

4. PC11T PILOT makes a mistake.

If course presentation ends in way 1, 2 or 3, PC11T PILOT saves file

USERRESPONSE and disconnects the learner from the DTSS. If PILOT detects an error (way 3), it emits an error message before exiting. APPENDIX A lists and explains the error messages. Special arangements may be made to have disconnection omitted when courses are being tested.

Following are two samples of PC11T PILOT error messages:

```
Error message  3:
  Error in reading PILOT course line.
At PILOT course line    1

Error message 11:
  No value expression in COMPUTE command.
At PILOT course line  544
  C:X=
```

The first message resulted from a bad executable course file. The second message resulted from a COMPUTE command which did not specify a value to be given to variable X. The course line is shown if possible.

If PASCAL detects a PILOT error (way 4), it emits a message like the following:

```
Subscript out of range.
Error occurred at line 1298 in procedure GETPOP
Called from line 2203 in program PILOT
```

If EXEC detects an error (way 4) and tells Pascal, messages like these may be seen:

```
I/O   TYPE 01 CODE 24 CONT 12      REENT ADR:  011757 BDI: 000004
PACKET ADR 047513

The lucid message above courtesy of 1100 Exec.
Error occurred at line 328 in procedure RRANREC
Called from line 337 in procedure RDIRREC
Called from line 478 in procedure RANSEQIN
Called from line 2186 in course PILOT
```

In the unlikely event that a Pascal message appears, please write it down and notify the CAI staff.

If PILOT terminates with a Pascal error message, the learner cannot enter ECL commands. If you try to enter an ECL command, the message "DATA IGNORED - IN CONTROL MODE" appears. To restore contact with EXEC enter:

```
@END
```

If you ever find PILOT not executing when you are still connected, restart the course or terminate your connection. To restart the course, enter:

```
@CAI
```

To terminate your connection, enter

@@TERM

2.2 Preparing and Presenting PC11P PILOT Programs

PC11P PILOT executes in IBM PCs and some microcomputers compatible with it. The file management, text editing and other features of the IBM Disk Operating System (DOS) were used to prepare PILOT programs. These features are well described in IBM Disk Operating System manuals and various other documents. IBM DOS and MSDOS are both products of Microsoft, Inc. and are nearly the same. So PC11P PILOT will probably operate properly in MS DOS systems.

This section briefly describes some DOS commands useful in preparing and presenting PC11P PILOT courses. For a fuller description see the DOS manuals.

Section 2.2.1 gives basic information on the DOS commands and files. Section 2.2.2 describes DOS commands pertinent to PC11P PILOT.

2.2.1 DOS Fundamentals

2.2.1.1 Starting DOS Processes

After DOS is started, when the system is waiting for you to tell it what to do, a prompt like this appears:

```
A>_
```

where the underscore is the prompt character. This says that the default disk drive is the A drive and that DOS is waiting for a command. To change the default drive to the C drive enter:

```
C:
```

The prompt changes to

```
C>_
```

When specifyng a file on a disk in a drive other than the default drive, prefix the drive letter followed by a colon to the file name like this:

```
A>B:WS
```

This entry might be used to start the Wordstar (WS) word processing program.

Four kinds of commands in an MS DOS system are internal commands, batch commands, external commands and executables. Examples are:

Example	Command Type
directory	internal
idr-ab.bat	batch
print.com	external
pilot.exe	external executable

The first column above gives the names of four commands. DIRECTORY (DIR

for short) is an internal command, that is, one built into DOS. The other three are contained in files which have the names listed. To start one of these, simply enter the part of its name to the left of the dot. PRINT and PILOT are external commands. A batch command is not really a command but a file containing one or more commands which DOS executes one at a time. For example, file IDR-AB.BAT contains one command:

```
PILOT  B:IDR  IDRHELP
```

which is used to start PC11P PILOT in a microcomputer which has just two disk drives: A and B.

External commands often require one or more file names or other information specifying what is to be done. For example, if you use the print command to print a file, you must specify the name of the file. For example:

```
PRINT  PP1E
```

tells DOS to print file PP1E in the current directory. If you do not specify required files, DOS will prompt you for them.

2.2.1.2 Naming and Creating Files

A PC11P PILOT program is built in a DOS text file containing a sequence of lines, each containing 80 characters or less. In DOS a file specifier consists of three items: a drive specifier, a filename and an extension. The drive specifier need not be entered if the file is in the current directory. A file specifier need not include an extension.

The DRIVE SPECIFIER is a letter followed by a colon. A FILENAME is from one to eight characters long. The characters can be a letter, a decimal digit or one of the following:

```
$ # & @ ! % ( ) - { } ' _ '
```

A filename EXTENSION is a period followed by one, two or three characters. Some extensions are meaningful to DOS, such as the BAT, COM and EXE extensions in examples above. BASIC program filenames have the extension BAS. Other extensions, like TXT, just tell people what kind of data is in the file.

There are various ways to build DOS text files. For example word processing packages like Wordstar may be used. The DOS line editor (EDLIN) can be used for quick creation or modification of DOS text files. To start EDLIN enter:

```
EDLIN <filespec>
```

where <filespec> is a file specifier. For example:

```
EDLIN B:COURSE2
```

If COURSE2 does not exist in the current B drive directory, a new file named COURSE2 is created there and the following message and prompt are displayed:

```
NEW FILE
*_
```

where the underscore is the cursor. EDLIN has two modes of operation: command mode and insert mode. When you see the prompt after starting EDLIN, you are in command mode. To enter text into a file, get into insert mode by entering the command:

```
I
```

Then type in lines of text, pressing the enter key at the end of each line. When finished inserting, return to command mode by pressing the Ctrl-Beak keys (hold down Ctrl, press Break, release both). To save the file, enter an E (End edit) command. To quit an editing session without saving changes made during it, enter Q (Quit edit).

In command mode you can delete, edit, insert and display lines. You can copy or move one or a sequence of lines. You can search for, delete, or replace text within one or more lines. You can transfer the content of a file into the file you are editing. This can speed insertion of frequently occurring sequences. For example the command

```
ttprc
```

can insert

```
T:
T:Press <RETURN> to continue.
A:
```

if file tprc contains this sequence.

The DOS editor keys facilitate editing. They are F1 through F5, Del, Esc and Ins.

2.2.2 PC11P PILOT Program Preparation and Presentation

As explained in section 2.1.2 above, a PC11 PILOT program is a file containing a sequence of PC11 PILOT commands. A PC11 PILOT course consists of a PC11 PILOT program which has a special lesson-section-unit (LSU) structure, and a related help file containing terms and their meanings. (The course structure is described in section 5.1 below.) Each of these two files exists two forms: an editable form and an executable form. An example of a PC11P course is "Introduction to DROLS Retrieval" (IDR), which is contained in files IDRE, IDRHELPE, IDR and IDRHELP.

Preparing a PC11 PILOT course includes preparing an editable course file and an editable help file, and inputting each file to its preprocessor, which outputs the executable file.

Presenting a PC11 PILOT course includes specifying which executable course and help files are to be used and starting the PC11 PILOT interpreter, which reads the files, presents the course, provides help, and otherwise interacts with the learner.

2.2.2.1 PC11P PILOT Program Preparation

A PC11P PILOT course in editable form is a standard DOS text file containing a sequence of commands written in the PC11 PILOT language. An example is IDR. An editable course file can be prepared on a word processor or minicomputer and communicated to an IBM PC. The DOS EDLIN line editor may be used to prepare or modify a course.

Preprocessor SEQRAN converts an editable course file into an executable course file. SEQRAN calls its input file SEQFILE and its output file RANFILE. To start SEQRAN enter, for example:

```
SEQRAN  PP1E  PP1
```

where PP1E (pilot program 1) is the editable file and PP1 is the executable file. If either file name is omitted, SEQRAN prompts for it:

```
C>SEQRAN
SEQFILE: PP1E
RANFILE: PP1
```

where PP1E and PP1 are responses to prompts.

A PC11P PILOT HELP FILE in editable form is a standard DOS text file containing terms and their meanings. Each term can have up to 60 characters, including spaces. An example is ACCESSION NUMBER. A start-of-header (SOH) character (ASCII code 1) must immediately follow the last character of each term. In EDLIN enter an SOH by typing Ctrl-V followed by A. The term's meaning begins on the next line. A blank line must follow the meaning. The terms must be in alphabetical order. An example is IDRHELPE, the help file for course IDR.

A PC11P help file can be prepared on a word processor or minicomputer and communicated to an IBM PC. The DOS EDLIN line editor can be used to prepare or modify a help file.

Preprocessor HSEQRAN converts an editable course file into an executable course file. HSEQRAN calls its input file HSEQFILE and its output file HRANFILE. To start HSEQRAN enter, for example:

```
HSEQRAN  PH1E  PH1
```

where PH1E (pilot help file 1 in editable form) is the editable file and PH1 is the executable file. If either file name is omitted, HSEQRAN prompts for it.

2.2.2.2 PC11P PILOT Program Presentation

When the PC11 PILOT interpreter is started, it reads the executable course and help files specified on the start line. To start PC11P PILOT enter, for example:

```
PILOT  PP1  PH1
```

If either file name is omitted, PC11P PILOT prompts for it. For example:

```
A:PILOT
PILOPROG: PP1
PILOHELP: PH1
```

where PP1 and PH1 are responses to prompts.

Presentation of PILOPROG ends in one of the following ways:

1. PC11P PILOT encounters an END command in the program when no subroutine is active (see section 4.3 below).

2. The learner enters the user command STOP. (See section 5.2.5 below.)

3. PC11P PILOT detects an error in the course.

4. PC11P PILOT makes a mistake.

Ways 1 and 2 are normal terminations. If PC11P PILOT detects an error (way 3), it emits an error message and exits. APPENDIX C lists and explains the error messages.

Following are two samples of PC11P PILOT error messages:

```
Error message  3:
  Error in reading PILOT program line.
At PILOT program line    1

Error message 11:
  No value expression in COMPUTE command.
At PILOT program line  544
  C:X=
```

The first message resulted from a bad executable course file. The second message resulted from a COMPUTE command which did not specify a value to be given to variable X. The course line is shown if possible.

In the unlikely event that PC11P makes a mistake, Pascal or DOS usually detects it and emits an error message. Please write it down and notify the CAI staff. If your PC fails to respond, reboot the system and restart PILOT.

3. THE PC11 PILOT LANGUAGE

Each PC11 PILOT program consists of a series of commands which tell the host computer system how to deliver some instruction. Each command is a line in a text file. Each line contains at least two and at most 80 characters. The characters are coded in the ASCII code.

3.1 Conventions Used in Describing PC11 PILOT Language

Following are some conventions we use in describing PC11 PILOT:

* One upper case letter is used for the PILOT command code.

* Tokens are used to represent parts of commands. Each token consists of one or more words enclosed by angle brackets and separated by underscore characters. Examples are <label> and <operation_code>.

* Items enclosed in square brackets are optional, as in [#]X.

* Exclamation characters are used to separate alternatives, as in [#!$].

3.2 Command Line Format

Each PILOT command is in one of the following formats:

```
[*<label> ]<operation_code>[<cond>]:
[*<label> ]<operation_code>[<cond>]:<operand>
```

Labels are optional in command lines. When a label is present, it is preceded by the character *, and begins in the second character position of the line. A label may appear on a line by itself:

```
*PART-TWO
```

When a labeled line contains a command, one space separates the label from the operation code. A label contains from 1 to 12 characters other than control characters and spaces. In comparing labels, lower and upper case characters are treated as different. A given character string may not be used to label more than one line in the same program. Labels are used as operands in commands which transfer control. UNIT LABELS are labels which consist of exactly four digits. They mark the start of a course unit. They are related to course structure and user commands (see section 5 below).

The operation code starts in character position 1 in unlabeled commands. Otherwise it starts after the space after the label. The operation code tells the interpreter what kind of action to take, such as outputting text to the user. There are nine PC11 PILOT command codes, and each is a single letter:

Code	Command	Function
A	ACCEPT	Take a response from the learner.

C	COMPUTE	Assign a value to a variable.
E	END	Terminate subroutine or course.
J	JUMP	Specify which PILOT program line to execute next.
M	MATCH	Look for string(s) in the learner's response.
R	REMARK	Comment for reader of editable PILOT program.
T	TYPE	Output text to learner.
U	USE	Call a subroutine.
X	ESCSTOP	Stop skipping PILOT program lines.

The first eight commands are standard. The X command is peculiar to PC11. The nine commands are explained in detail in section 4 below.

An execution condition may be specified immediately after the operation code. The character : immediately follows the operation code, if no execution condition is present. Otherwise it immediately follows the condition.

The execution condition may be the letter N or the letter Y or an expression in parentheses. N and Y are mutually exclusive, but each may be combined with an expression. These alternatives may by symbolized as follows:

<cond>=N!Y!(expression):N(expression)!Y(expression)

N and Y are short for the expressions &N<>0 and &Y<>0, respectively, and are related to MATCH commands. An execution condition is an arithmetic or Boolean expression. When a command containing an execution condition is processed, the expression is evaluated and the action is taken if and only if the expression is true. Expressions and their evaluation are discussed in section 3.4 below.

The operand field is required or optional depending on the operation code. The ACCEPT command usually has no operand. The END command never has one. The JUMP and USE commands must have one. The TYPE command may or may not have one.

3.3 Variables

PC11 PILOT has a small set of variables for saving values. Two types of variables are supported: numeric variables and character string variables. Arrays are not supported.

3.3.1 Numeric Variables.

Two sets of numeric variables are available: system numeric variables and application numeric variables. Each set contains one variable for each letter of the alphabet. Upper and lower case letters in numeric variable names are treated the same. Each numeric variable name consists of a type indicator followed by a letter. The system variable names are &A through &Z, and the application variable names are #A through #Z. The prefix # may be omitted in expressions.

All numeric variable values are stored as single precision real values. In the Univac 1100 version of PC11 PILOT the magnitude can be between 10**38

and 10**-38, with a significance of approximately 8 digits. In the IBM PC version, the maximum value is about 1.7E+38, with about 7 digits of precision.

All numeric variables are set to zero when execution of the PC11 Pilot interpreter starts. The value of a numeric variable may be altered by an ACCEPT command or a COMPUTE command. When an ACCEPT command containing a numeric variable name in its operand field is executed, the numeric variable is set to the value of the first numeric expression in the input from the user. When a COMPUTE command contains a numeric variable name at the start of the operand field, the variable is set to the value of the expression following the equal sign (=). Examples are:

```
A:#R
C:p=q*r
```

When a TYPE command contains a numeric variable name in its operand field, the value of the variable is rounded up, converted to an integer string and output in place of the variable name. For example, the command sequence

```
C:q=12.3
C:r=45.6
C:p=q*r
T:#P is the answer.
```

will output "561 is the answer."

3.3.2 System Numeric Variables

The values of some system variables are set by the PILOT Interpreter. It is best not to alter these values. The description of each PILOT command in section 4 below tells which system variables (if any) are altered by the command and how. Following is a summary:

&A is set by each ACCEPT commmand to the number of times in a row the particular command (PILOT program line) has been executed.

&C is set to the value (true or false) of the execution condition expression in the most recently executed statement containing such a condition.

&L is set by each ACCEPT command to the length of the response received, not counting the return character and any trailing blanks.

&M is set to zero by each ACCEPT command and increased by 1 by each MATCH command which finds a match.

&N is set by each MATCH command to false (zero) if a match is found, and to -1 (true) if not.

&X is set to 0 by each ESCSTOP command. When &X is false (0), both "T:Press <RETURN> to continue." commands and ESCSTOP commands stop escape skipping. When &X is true (nonzero), only an ESCSTOP command stops escape skipping.

&Y is set by each MATCH command to the position number in its operand field of the first matching operand if a match with an item in the most recent user response is found, and to zero if not.

3.3.3 String Variables

String variables are used to save and output character strings. Each string variable name consists of the type indicator $ followed by 1 to 12 characters from the set containing A-Z, a-z, 0-9, dash (-) and underscore (_). In string variable names, upper case letters are different from lower case letters. The value of a string variable is a string of from 0 to 80 characters.

String variables are used to save character strings received by the ACCEPT command. String variables are also set by the COMPUTE commands. Example:

```
C:TEAM1=BIRDS
C:TEAM2='TIGERS'
```

In the second example, the apostrophes are not part of the string.

When a TYPE command containing a string variable name in its operand field is executed, the variable's value replaces the variable name in the output. If, however, the string variable has not been given a value during this execution of the PILOT interpreter, the variable name itself is output.

PC11 initializes variable $NAME to Learner and variable $LOOP to LOOP (for use with J:@A commands).

3.4 Expressions in PC11 PILOT

In computing languages, EXPRESSIONS are constructs denoting rules of computation for obtaining values. Although expressions in PC11 PILOT are similar to expressions in other programming languages, what is said about expressions in what follows is meant to apply only to the former.

3.4.1 Uses of Expressions

Expressions may appear in PC11 PILOT commands in two places: in the execution condition field of any command, and in the operand field of certain ACCEPT and COMPUTE commands. Examples are:

```
C:G=3
C:P=Q+R
T(G<1):No more guesses.
A:#X
```

Here 3, Q+R and G<1 are expressions. The first command gives the value 3 to the variable G. The second command computes the value of Q+R and gives it to variable P. The third command is executed if and only if the value of the expression G<1 is true, that is, if and only if G is less than 1. The fourth command does not treat #A as an expression but as a variable which is to receive the value of an expression to be entered by the user. The command prompts for a response from the user, evaluates the first expression in the response, and gives the value to #X.

3.4.2 Components of Expressions

Expressions are composed of operands, operators and parentheses. A numeric constant, such as 1.5, or a numeric variable name by itself is a simple expression. Before any expression is evaluated, all spaces are removed from it.

The value of an expression may be Boolean (true or false) or numeric. If the value of an execution condition expression is numeric, it is converted to Boolean. Zero becomes FALSE and any other value becomes TRUE. If the value of an expression to be given to a numeric variable is Boolean, it is converted to numeric. FALSE becomes 0 and TRUE becomes -1.

Operands in expressions are constants, numeric variables or expressions. For example in the expression A+(5.7*&B), 5.7 is a constant, A and &B are numeric variables, + and * are operators, and 5.7*&B is an expression. Operands may be numeric or Boolean. Numeric operands have real (floating point) values. Boolean operands have one of two values: TRUE or FALSE.

Operators may be divided in three ways: by number of operands, by type and by precedence.

Operators are divided by NUMBER OF OPERATORS into unary or binary. A UNARY OPERATOR is one which has only one operand, such as the - operator in -7. The unary operators are +, -, NOT and BNOT. BNOT means "bitwise NOT", that is, each bit in the operand is changed from 0 to 1 or from 1 to 0. The BNOT operator converts the operand to an integer, changes each 0 bit in it to 1 and each 1 bit to 0, and converts the resultant integer value to real.

A BINARY OPERATOR is one which has two operands, such as the operator * in the expression a*b. The symbols + and - (plus and minus) are used both as unary and as binary operators. The way they are used in a particular case depends on the context. (Other symbols, such as ~ and ^ could have been used for unary plus and minus.) Unary + is an identity operator. Unary - reverses the sign of the operand.

Operators are divided by TYPE into numeric, relational and Boolean. The NUMERIC OPERATORS are unary +, - and BNOT, and the following:

Operator	Meaning
*	multiplication
/	division
//	integer division
+	addition
-	subtraction
BAND	bitwise AND
BOR	bitwise OR
BXOR	bitwise exclusive OR

Integer division here means dividing one real number by another, converting the result to an integer (fraction is dropped), and converting the integer to a real value. Bitwise binary operations convert each operand to an integer, perform an AND, OR or XOR on corresponding bit pairs, and convert

the result to a real value.

The RELATIONAL OPERATORS are =, <>, <, <=, > and >=, meaning equal, not equal, less than, less than or equal, greater than, and greater than or equal.

The BOOLEAN OPERATORS are NOT, AND and OR.

Examples are:

expression	operator	type
x+y+z	+	numeric
f < g+h	<	relational
a<b OR c<d	OR	Boolean

The operand(s) of each numeric or relational operator must be numeric. The operand(s) of each Boolean operator must be Boolean.

Operators are divided by PRECEDENCE into seven groups. If an expression contains one or more pairs of parentheses, the expression inside each parenthesis is evaluated before those outside it. In expressions containing no parenthesis, operators with higher precedence are executed before those with lower precedence. Sequences of operators with the same precedence are executed from left to right. The unary numeric operators +, - and BNOT have the highest precedence. The Boolean operator OR has the lowest. The following table shows the precedence and other characteristics of each group.

operator group	precedence	number of operands	operator type	operand type	result type
+ - BNOT	7	1	numeric	numeric	numeric
* / //	6	2	numeric	numeric	numeric
+ - BAND	5	2	numeric	numeric	numeric
BOR BXOR	5	2	numeric	numeric	numeric
= <> <	4	2	relational	numeric	Boolean
<= > >=	4	2	relational	numeric	Boolean
NOT	3	1	Boolean	Boolean	Boolean
AND	2	2	Boolean	Boolean	Boolean
OR	1	2	Boolean	Boolean	Boolean

Note that relational operators input numeric values and output Booleans. This is the only way Booleans arise in PC11 PILOT. There are no Boolean variables in which to save Boolean values. However, if a COMPUTE command specifies that a numeric variable is to be given a Boolean value, that value is converted from true or false to -1 or 0, respectively. These can be converted back to Boolean by using a relational expression, as in the following example:

```
C:V=(7*X*Y)<L
T(V<>0):The product is under the limit.
```

Here V<>0 would be equivalent to V=TRUE, if the latter were an expression in PC11 PILOT.

Parentheses may be used in expressions for clarity or to override the left-to-right or precedence rules. Following are illustrations of precedence:

expression	equivalent expression
a*b+c	(a*b)+c
a+b*c	a+(b*c)
NOT a=1 AND b=2	(NOT a=1) AND b=2
p=1 OR q=1 AND r=2	p=1 OR (q=1 AND r=2)

4. PC11 PILOT COMMAND DESCRIPTIONS

This section describes each of the nine PC11 PILOT commands. For each command the following are given: meaning, syntax, description, example(s), system variable(s) altered and notes.

4.1 ACCEPT Command (A)

Meaning: Accept input from the user or a string variable

Syntax:

```
[*<label>]A[<cond>]:                          {type A1}
[*<label>]A[<cond>]:<numeric_variable>        {type A2}
[*<label>]A[<cond>]:<string_variable>         {type A3}
[*<label>]A[<cond>]:@<string_variable>        {type A4}
```

Description:

A type A1 command writes the prompt character to the user, waits for a response, inputs it, simplifies it, and puts it into the accept buffer with one space before and after it. The simplification consists of removing trailing blanks and converting lower case letters to upper case.

Type A2 and A3 commands have the same effect as type A1, except that each sets a variable according to the user's response. A type A2 command sets a numeric variable to the numeric value of the first expression in the response. The expression begins with the first character in the set:

(+ - 0 1 2 3 4 5 6 7 8 9

and ends with the end of the accept buffer. A type A3 command sets a string variable to the user's response without trailing blanks.

A type A4 command does not prompt or input a user response but copies the value of the specified character string variable into the accept buffer.

Examples:

```
A:
A:#C
A:$LAST_NAME
A:@$TEST-RESPONS
```

Example 1 is a type A1 command, which outputs the prompt character, inputs a user response, simplifies it and puts it between two spaces in the accept buffer.

Example 2 is a type A2 command, which puts the response in the accept buffer and sets the value of system numeric variable &C to the numeric value of the first expression in the response.

Example 3 is a type A3 command, which puts the response in the accept buffer and sets the value of string variable $LAST_NAME to the user response.

Example 4 is a type A4 command, which does not solicit or accept a user response but copies the value of string variable TEST-RESPONS into the accept buffer, putting a space before and after it.

System variables altered:

&A is set to the number of times in a row this accept command (PILOT program line) has been executed. Only executions of accept commands are used in determining the value of &A.

&C is set to the value of the execution condition, if the ACCEPT has one.

&E is set to -1 (true) if the ACCEPT type is A2 and the response contains a numeric expression. Otherwise, it is set to false (0).

&L is set to the number of characters in the response, not counting trailing blanks, return characters, or the spaces put in the accept buffer before and after the response.

&M is the successful match counter and is set to zero.

Notes:

1. Lower case letters in the response become upper case in the accept buffer.

2. User responses are truncated after 78 characters.

3. Type A4 commands may be used to test a PILOT program's reaction to various user responses.

4. Special responses called user commands are recognized by the PC11 PILOT interpreter. These are described in section 5.2.

4.2 COMPUTE command (C)

Meaning: Set a numeric variable to the value of a numeric or Boolean expression, or set a string variable to the user's response.

Syntax:

```
[*<label>]C[<cond>]:<numeric_variable>=<expression>          (type C1)
[*<label>]C[<cond>]:<string_variable>=<string>!'<string>'    (type C2)
```

Description:

A type C1 command sets the value of the numeric variable specified in the operand to the numeric value of the arithmetic or Boolean expression following the equal sign (=).

A type C2 command sets the value of the string variable specified in the operand to the character string following the equal sign (=). The string may put between single quote marks (') if these are the character after the equal sign and the last character in the command. If no character follows the equal sign, the string variable is set to the empty string.

Examples:

```
R:R is the number of correct answers.
R:W is the number of wrong answers.
R:Compute the average and present it to the user.
C:R=21
C:W=27
C:$U=quiz
C:A=(100*R)/(R+W)
T:Your average for this $U is #A%.
```

System variables altered:

&C is set to the value of the execution condition, if the COMPUTE command has one.

Notes:

1. Variables are explained in section 3.3.

2. Expressions are explained in section 3.4.

4.3 END Command (E)

Meaning: END subroutine or program.

Syntax:

[*<label>]E[<cond>]:

Description:

When an END command is executed, the return line stack is tested. If it is empty, the PILOT program terminates. If it is not empty, the line whose number is on top of the stack is executed next. This is the line after the most recently executed USE command. The END command decreases the return line stack top pointer by one.

Example:

```
T:Do you wish to see the list of topics?
A:
M: Y,
UY:SHOW-T-LIST
. . . .
R:End of program
E:
. . . .
*SHOW-T-LIST
T:    GENE SPLICING
. . . .
R:End of subroutine.
E:
```

System variables altered:

&C is set to the value of the execution condition, if the END command has one.

Notes:

1. Subroutine calls and returns are explained in section 4.8, which is about the USE command.

4.4 JUMP Command (J)

Meaning: Jump to a labeled line in the PILOT program being executed.

Syntax:

```
[*<label>]J[<cond>]:[*]<target_label>      (type J1)
[*<label>]J[<cond>]:@A                     (type J2)
[*<label>]J[<cond>]:@M                     (type J3)
```

Description:

In a type J1 command, <target_label> is a statement label with or without a preceding asterisk. When a type J1 command is executed, the line executed next is the target label's line. If the line contains a command, the command is executed. If it does not, the first command following the label's line is executed. If the label does not exist, the PILOT interpreter emits an error message and terminates.

When a type J2 command is executed, the line executed next is the most recently executed ACCEPT command.

When a type J3 command is executed, the line executed next is the next following MATCH command.

Examples:

```
T:Do you wish to continue?
A:
M: N ,
JN:@M
T:Bye!
E:
M: Y ,
JY:GO
T:Enter Y for Yes or N for No.
J:@A
*GO T:On you go!
```

This sequence prompts the learner and takes a response. If the response is the letter N, "Bye!" is output to the user and the program (or subroutine) terminates. If the response is the letter Y, "Here you go!" is output to the user and the program continues. If the response is not M or Y, "Enter Y for Yes or N for No." is output to the user, and the ACCEPT command is executed again. The program cycles until the learner enters an N or a Y (1 character).

System variables altered:

&C is set to the value of the execution condition, if the JUMP has one.

Notes:

1. Using a type J1 or J2 command is sometimes more convenient than using a label.

2. The five-character MATCH command M: , can be used instead of a label as the target of the forward jumping command J:@M. An M: , command affects no system variable. Of course there must be no MATCH command between the J:@M and the M: command. The M: , command affects no system variable. The two-character command M: is a fatal error.

3. An ACCEPT command of the form A:@$<string_variable> can be used instead of a label as the target of the backward jumping command J:@A, if the string variable exists, that is, has been given a value (usually by a COMPUTE command). The command A:@$LOOP is convenient, since it uses a string variable initialized to LOOP by the PC11 PILOT interpreter. Of course there must be no ACCEPT command between the J:@A and the A:@$ commands. The A:@$ command affects the accept buffer and some system variables.

4.5 MATCH Command (M)

Meaning: Compare each operand with the content of the accept buffer to determine if any operand is in the buffer.

Syntax:

```
[*<label>]M[<cond>]:<string>!<string_variable>[,...]          (type M1)
[*<label>]M[<cond>]:,<s><string>!<string_variable>[<s>...]    (type M2)
```

Description:

Each MATCH command has one or more operands. In type M1 commands successive operands are separated by a comma. In type M2 commands the operand field starts with a comma followed immediately by the character to be used as a separator for this command. This is needed if a match operand contains a comma.

The MATCH command takes its operands one at a time, from left to right, and compares them with the content of the accept buffer. Each operand is a string constant or the value of a string variable. If an operand matches a string in the buffer, the "no match" system variable &N is set to false (zero), the match pointer &Y is set to the position number in the operand list of the first matching operand, if any, and the match counter &M is increased by 1. If no operand is found in the accept buffer, &N is set to -1 (true), &Y is set to false (0), and &M is set to 0.

Examples:

```
R:Start of program
T:Please enter your identifier.
A:$IDENTIFIER
M: SMITH , JONES , BROWN ,
T(&Y=1):Hello Steve.
T(&Y=2):Hello Jean.
T(&Y=3):Hello Bobby.
TN:$IDENTIFIER is not a valid identifier.
EN:
....
```

If the learner's identifier is not one of the three listed, the END command terminates the program.

The MATCH command sets system variables &N and &Y as follows:

command	response	&N	&Y
M: SMITH , JONES , BROWN ,	SMITH	0	1
M: SMITH , JONES , BROWN ,	JONES	0	2
M: SMITH , JONES , BROWN ,	BROWN	0	3
M: SMITH , JONES , BROWN ,	GREEN	-1	0

If the user's response contains one or more of the three names in the MATCH operand list, &N is set to 0 and &Y is set the lowest position number in

the list of any matching response item. Thus if the user enters BROWN JONES SMITH, the MATCH will set &Y to 3. If no operand item is in the response, &N is set to -1 (true) and &Y set to false (0).

Thus if JONES is entered, &Y is set to 2 and the second TYPE command outputs "Hello Jean" to the user. If any of the three listed names is entered, the TN and EN commands are not executed. If the response contains none of the MATCH operands, the TN command outputs its message and the EN command terminates the program. If the user enters Green, the TN outputs "GREEN is not a valid identifier."

The ACCEPT command puts a space in the accept buffer before and after the response. The following example shows the effect of spaces in MATCH command operand lists:

```
R: Suppose a learner's response to a prompt is "warm" (4 letters),
R: so that the accept buffer contains " WARM " (six characters).
R: If the following commands are then executed, which ones will NOT set
R: the match pointer &Y to 1?
R:
*1 M:WARM
*2 M: WARM
*3 M:WARM ,
*4 M: WARM ,
*5 M:WAR
*6 M: WAR
*7 M:WAR ,
*8 M: WAR ,
*9 M:ARM
*10 M: ARM
*11 M:ARM ,
*12 M: ARM ,
```

See note 5 below for the answer.

System variables altered:

&C is set to the value of the execution condition, if the MATCH command has one.

&M is increased by 1 if a match is found.

&N is set to 0 (false) if a match is found, and to -1 (true) if not.

&Y is set to the position number in the operand list of the first matching operand if a match is found, and to 0 if not.

Notes:

1. The ACCEPT command puts a space in the accept buffer before and after the user response.

2. Put a separator after the last character of the last MATCH operand. IF no separator follows the last operand, trailing spaces in the operand are removed before the MATCH is executed.

3. Lower case letters in MATCH command operands are changed to upper case before comparing begins.

4. The five-character MATCH command M: , can be used instead of a label as the target of the forward jumping command J:@M. An M: , command affects no system variable. Of course there must be no MATCH command between the J:@M and the M: command. The M: , command affects no system variable. The two-character command M: is a fatal error.

5. The answer to the question in the "WARM" example is: 7, 8, 10 and 12.

4.6 REMARK Line (R)

Meaning: The operand field contains a remark.

Syntax:

[*<label>]R:<character_string>

Description:

REMARK is has the form of a command but is not a command and is not executed. REMARK lines are used to tell things to people reading a listing of the editable form of the program. A REMARK line may appear anywhere in a program.

Example:

R:Variable #T holds the number of tries.

System variables altered: None.

Notes:

1. The SEQRAN preprocessor outputs R: (two characters) instead of nonempty REMARK statements. This reduces the size of the output file without changing line numbers, so that the interpreters's error messages can give the line number of the erroneous line in the editable PILOT program file.

2. Labels on REMARK lines are permitted. Execution conditions in REMARK statements are permitted but serve no purpose.

4.7 TYPE Command (T)

Meaning: Type text to the user.

Syntax:

[*<label>]T[<cond>]:[<string>!<variable>] . . . [;]

Description:

The TYPE command is used to output text to the user. The text to be displayed is specified in the operand field. There may be zero or more operands. The pieces of text specified by operands are output with no character between them. Each specifier in the operand field is a string constant or a variable name.

String constants are output without change, except that "$$", "&&" and "##" are changed to "$", "&" and "#", respectively. These characters are doubled in TYPE operand string constants to make it possible to distinguish between variables and constants.

If a specifier is a string variable name, the value of the variable (a string) is output. If the string variable has not received a value during the execution of the program, the name of the variable, preceded by a $ character is output. If a specifier is a numeric variable, value of the variable is rounded up, converted to an integer string and output.

After all specified characters have been output, a return character is output to position to the next line. If there is no operand, just the return is output. The return is omitted if the last character in the operand is a semicolon (;).

Examples:

```
R:Get the user's state of being and echo it.
T:How are you today?
A:$HOW-YOU
T:I'm $HOW-YOU too.

R:Keep price in a string variable.
C:$P='349.95'
R:Keep item number, month, day and year in separate numeric
R:variables.
C:I=29007
C:D=31
C:M=12
C:Y=86
R:Output item number, price and date.
T:The price of item #I is $$$P (plus tax).
T:The offer is good only through ;
T:#M/#D/#Y.
```

The output of the second example is:

```
The price of item 29007 is $349.95.
```

The offer is good only through 12/31/86.

System Variables Altered:

&C is set to the value of the execution condition, if the TYPE command has one.

Notes:

1. The values of all numeric variables are kept as real values. The way these values are rounded up for output by the TYPE command depends on the sign. For positive values, the rounded value is the greatest integer less than or equal to the sum of the real value and 0.5. For negative values, the rounded value is the least integer greater than or equal to the sum of the real value and -0.5.

4.8 USE Command (U)

Meaning: Use a subroutine.

Syntax:

```
[*<label>]U[<cond>]:[*]<target_label>     (type U1)
[*<label>]U[<cond>]:@A                    (type U2)
[*<label>]U[<cond>]:@M                    (type U3)
```

Description:

The USE command specifies which line in the PILOT program is to be executed next and provides for a return to the line immediately following itself.

In a type U1 command, <target_label> is a statement label with or without a preceding asterisk. When a type U1 command is executed, the line executed next is the target label's line. If the line contains a command, the command is executed. If it does not, the first command following the label's line is executed. If the label does not exist, the PILOT program emits an error message and terminates.

When a type U2 command is executed, the line executed next is the most recently executed ACCEPT command.

When a type U3 command is executed, the line executed next is the next following MATCH command.

The USE command increases the return line stack top pointer by one and puts the number of the line after itself on top of the stack. Thus the USE statement does a subroutine call. The sequence of statements from the label which is the target of the USE command and the END command which returns control to the line after the USE is a subroutine. Subroutines may be nested, that is, one subroutine may contain a USE command calling another subroutine.

When an END command is executed, the return line stack is tested. If it is empty, the program terminates. If it is not empty, the line whose number is on top of the stack is executed next. This is the line after the most recently executed USE command. The END command decreases the return line stack top pointer by one.

Example:

```
*START U:FIRST
T:command.
E:
*FIRST U:SECOND
T:handy ;
E:
*SECOND U:THIRD
T:is a ;
E:
*THIRD
T:U ;
```

E:

This example is a subroutine or a program, depending on the context.

System variables altered:

&C is set to the value of the execution condition, if the USE command has one.

Notes:

1. The USE and JUMP commands have the same effects, except that USE provides for a return and thus affects the next executed END command.

2. A subroutine may contain JUMP commands, but it is good practice not to jump into or out of a subroutine. Instead enter at the subroutine's first line and exit through the END command which is its last line.

3. The return line stack can hold ten line numbers, so no more than 10 USE commands may be executed without an END being executed.

4. The five-character MATCH command M: , can be used instead of a label as the target of the forward jumping command U:@M. The M: , command affects no system variable. Of course there must be no MATCH command between the U:@M and the M: commands. The M: , command affects no system variable. The command M: is a fatal error.

5. An ACCEPT command of the form A:@$<string_variable> can be used instead of a label as the target of the backward jumping command U:@A, if the string variable has been given a value (usually by a COMPUTE command). The command A:@$LOOP is convenient, since it uses a string variable initialized to LOOP by the PC11 PILOT interpreter. Of course there must be no ACCEPT command between the U:@A and the A:@$ commands. The A:@$ command affects the accept buffer and some system variables.

4.9 ESCSTOP command (X)

Meaning: Stop line skipping started by the response ESC.

Syntax:

[*<label>]X[<cond>]:

Description:

If escape skipping is in progress, the escape indicator is turned off. Otherwise nothing is done.

The three-letter response ESC is a user command recognized by any ACCEPT command. It starts escape skipping by turning on the escape indicator. This suppresses execution of PILOT program commands until either of two conditions is fulfilled: (1) an ESCSTOP command is met, or (2) the value of system variable &X is false (not zero) when the following command is met:

```
T:Press <RETURN> to continue.
```

In the second case, the "Press <RETURN>" line and the line after it (usually A:) are also skipped. In either case, &X is set to 0 and the escape indicator is turned off.

The purpose of the ESC command is to permit the learner to break free from a question or exercise. The ESCSTOP command is used to control how much is skipped.

Example:

```
C:&X=1
T:What command displays all the directory paths on a drive?
A:
M: TREE,
JN:@A
T:Good $NAME!
T:
T:Press <RETURN> to continue.
A:
T:What command displays the names of the files in all the
T:directory paths on the default drive?
T:
A:
M: TREE/F,
JN:@A
T:Very good $NAME!
T:
T:Press <RETURN to continue.
A:
X:
T:Let's turn to another topic.
```

This example has two ACCEPT-MATCH-JUMP sequences each of which repeats until the correct answer or ESC is entered. Since &X is made true by the

first command, if ESC is entered, skipping won't stop until the "Lets turn" line.

System variables altered:

&C is set to the value of the execution condition, if the COMPUTE command has one.

&X is set to 0 (false).

Notes:

1. Since no JUMP, USE, or END (or other) command is executed during escape skipping, the interpreter proceeds through the PILOT program examining each line until an escape-stopping condition is fulfilled or the end of the program is reached.

2. The escape indicator is a Boolean variable in the interpreter. Only one thing turns it on: the response ESC to an ACCEPT command. Only two things turn it off: (1) an ESCSTOP command or (2) a "T:Press <RETURN> to continue." command when system variable &X is false (0).

3. User commands are described in section 5.2 below.

5. PC11 PILOT COURSE STRUCTURE AND USER COMMANDS

THE PC11 PILOT interpreter recognizes nine user commands. Some of these commands presuppose that the PC11 PILOT program being presented is a PC11 PILOT course. Section 5.1 describes the PC11 PILOT course structure. Section 5.2 describes the PC11 PILOT user commands.

5.1 PC11 PILOT Course Structure

A PC11 PILOT COURSE is built of lessons which are built of sections which are built of units. This lesson-section-unit (LSU) structure distinguishes courses from other PC11 PILOT programs. Each UNIT in a course starts with a unit header line which has the following format:

```
*<un> T:U<un> <unit heading>
```

where <un> is a unit number consisting of exactly four decimal digits and <unit heading> is a descriptive title containing up to 66 characters. Thus each unit header line is a labeled TYPE command. For example:

```
*0220 T:U0220 OVERVIEW OF DTIC
```

The label 0220 is a unit label. In each PC11 PILOT course, the unit labels must be in ascending sequence. If not, the SEQRAN preprocessor catches the error and terminates abnormally.

In each unit number, the first two digits are the lesson number. The next digit is the section number relative to the lesson, and the last digit is the unit number relative to the section. Thus unit 1234 is unit 4 of section 3 of lesson 12.

Lesson 0 of each course introduces the learner to the PC11 PILOT CAI system.

The first unit number in each lesson ends in 00. For example:

```
*1100 T:U1100 DROLS SORT COMMANDS
```

The first unit in a lesson usually gives an overview of the lesson. The first unit number in a section ends in 0.

5.2 PC11 PILOT User Commands

PC11 PILOT provides nine user commands. They give the learner some control over the presentation, offer assistance, and permit user feedback. The user commands are:

SHOW	NOTE	ESC
GOTO	TIME	
BACK	WAIT	
HELP or ?		
STOP		

NOTE, TIME and WAIT are available in the dial-up version but not in the microcomputer version of PC11 PILOT.

User commands other than ESC are recognized when and only when the following prompt appears:

Press <RETURN> to continue.

5.2.1 The SHOW User Command

The SHOW user command lists all or a subset of the unit headings in the course. Such a listing is, in a way, an outline of the course. You may show all headings, lesson headings only (top level), lesson and section headings (top two levels), or some other subset of headings.

The question mark (?) can be used in show command unit numbers. It is a "wild card" meaning any digit. Thus 123? means 1230 through 1239.

Examples are:

Command	Headings shown
SHOW	depends on answers to prompts
SHOW ????	all in course
SHOW 0021	one
SHOW 10??	all in lesson 10
SHOW 123?	all in section 3 of lesson 12
SHOW 0900 1199	all in lessons 9, 10 and 11

If a subset of unit numbers is specified, it may contain many numbers which are not unit labels in the course being presented.

5.2.2 The GOTO User Command

The GOTO user command lets the user to alter the order in which course units are presented. For example:

GOTO 0021

causes unit 0021 to be presented next. The unit number must have exactly four digits. Thus the following commands go nowhere:

GOTO 333 GOTO 55555

If there is a problem with the unit number part of the command, the learner is helped.

5.2.3 The BACK User Command

The BACK user command returns to a previously presented part of the course. BACK or BACK 1 returns to the line after the most recently presented "T:Press <RETURN> to continue." (PRC) line. BACK 2 returns to the line after the second most recent PRC line, and so on.

The GOTO command erases the PRC line trail, so that BACK cannot return beyond the unit gone to.

5.2.4 The HELP User Command

The HELP user command is just the word HELP or a question mark (?) alone on a line. The prompt character becomes '?' when help starts. It becomes '>' again when the learner quits getting help.

The HELP feature is primarily a glossary with an index to it. These permit quick recall of meanings of some terms related to the course subject or the PC11 PILOT system. You can also view the page (window) of the index which contains (or would contain) a given term.

The HELP feature is self explanatory. Try it.

5.2.5 The STOP User Command

The STOP user command lets the learner terminate the course. When STOP is entered, the learner is asked:

DO YOU WISH TO TERMINATE THIS COURSE (Y/N)?

to make sure there's no mistake. If the answers is Y, the course is terminated. If it is N, the 'Press <RETURN>' prompt reappears. If you plan to stop and continue the course later, write down the current unit number. Then you can skip the preceding material next time (using the GOTO command).

5.2.6 The NOTE User Command

The NOTE user command is available only in the dial-up PC11 PILOT system.

The NOTE user command enables you to comment on the course or the course presentation system or other topic. You are prompted to enter lines of text. Questions and other types of input are accepted. The lines entered are placed in the session's user response file and can be read by the training staff after your the session ends.

You are encouraged to comment and question freely. Notes can lead to course improvement and removal of user difficulties.

Put short notes on the command line like this:

NOTE This part is confusing.

To make a long note, enter only:

NOTE

and you'll be prompted for lines until you enter an empty line.

5.2.7 The TIME User Command

The TIME command is available only in the dial-up PC11 PILOT system.

To obtain the date and time from the dial-up computer enter:

TIME

5.2.8 The WAIT User Command

The WAIT command is available only in the dial-up PC11 PILOT system.

If you do not respond to any prompt within a few minutes, you receive the following message:

TIMEOUT WARNING

Then, if you do not respond within another few minutes, your terminal will be disconnected, so that another dialer can use the communication port. But the wait command resets the timeout timer.

Suppose you anticipate disconnection but are not quite ready to continue with the course. Enter the WAIT command to avoid disconnection.

5.2.9 The ESC User Command

The three letter response ESC is recognized by every ACCEPT command. The purpose of this user command is to permit the user to break free from a question or exercise.

ESC starts escape skipping by turning on the escape indicator. This causes execution of PILOT program lines to be suppressed until (1) an ESCSTOP (X) command is met or (2) the value of system variable &X is false (0) when this PILOT line is met:

T:Press <RETURN> to continue.

In the second case, the "Press <RETURN>" line and the following line (usually A:) are also skipped. In either case &X is set to 0 (false), and the escape indicator is turned off. ESCSTOP commands and "Press <RETURN>" lines control how much is skipped.

APPENDIX A

HOW TO USE THE DTIC CAI COURSE "INTRODUCTION TO DROLS RETRIEVAL" IN THE DTIC TIME SHARING SERVICE (DTSS)

PLEASE READ ALL INSTRUCTIONS BEFORE SIGNING ON

1. A computer-assisted instruction (CAI) system which delivers "Introduction to DROLS Retrieval" is in the DTIC ADPE Time Sharing System (DTSS). This system is hosted by the DTIC Univac 1100/61 computer. To be able to take the course you must sign onto this system. This requires you to have a DTSS site identifier (6 characters), user identifier (6 characters) and password (up to 6 characters). If your organization is registered to use the DTSS, obtain the sign-on items from the appropriate person. If your organization is not registered to use the DTSS, you may obtain sign-on items for temporary guest use of the DTSS for CAI only.

2. In the following description of the sign-on process, we refer to some terminal keyboard keys and some characters. The symbols [control], [return] and [backspace] represent keys. The return and backspace keys correspond to the RETURN and BACKSPACE characters, which are represented as numbers in computers. The control key is used with other keys to enter special characters. For example, to enter the CANCEL character, hold down the control key, press and release the X key, then release the control key. In other words, "Enter [control]X."

3. When you are ready to take a CAI course, you must first sign onto the DTSS.

a. Check your terminal's characteristics. It must operate at 300 baud, in the half-duplex mode, with even parity or none. If your terminal has switches for these parameters, set them.

b. Dial 274-8401 or 274-8574 for access by commercial lines. If you have an acoustic coupler modem, listen for the high-pitched carrier tone. When you hear it, place the telephone handset into the coupler. When the carrier light on your modem or terminal comes on, the connection has been made. If you receive a busy signal or no answer, try the other number. If this fails, try again later.

c. Enter your site identifier. No RETURN is required. The DTSS should respond within 20 seconds with the following prompt:

```
ENTER USERID/PASSWORD:
>
```

d. Enter your USERID, a slash (/), and your password immediately after the prompt character (>). For example:

```
>XYZ12A/OPENUP[return]
```

If the USERID you enter is unacceptable, the system says "ID NOT ACCEPTED" and disconnects you. If this happens, hang up the phone and go back to step a. If the USERID is acceptable but the password is wrong for it, the

system repeats the prompt of step c. If this happens, go back to step d.

If you make a mistake in entering your USERID/PASSWORD and realize you did before you press [return], DO NOT BACKSPACE. Instead, enter a CANCEL character, as explained in item 2 above. This tells the DTSS to discard all you entered after the last prompt. It is like backspacing to the prompt. The DTSS computer does not act on the BACKSPACE character until after step f. So, if you enter a BACKSPACE character before that, it will be part of your USERID/PASSWORD entry and make it unacceptable. You may use the CANCEL character at any time in your session to discard a line you are entering. When you do this, the DTSS then sends you a RETURN and a LINE FEED character but not a prompt.

e. After you sign on, something like the following appears:

```
*DESTROY USERID/PASSWORD ENTRY
*UNIVAC 1100 OPERATING SYSTEM LEV.  37R2C*DTIC87(RSI)*
**********************************************************

***  USERS OF THIS TERMINAL HAVE THE RESPONSIBILITY   ***

***  TO RESTRICT USE TO AUTHORIZED PERSONNEL ONLY     ***

**********************************************************

RUN NUMBER 63

LAST RUN AT:  100783 091712
DATE:  101483      TIME:  075048
```

4. The CAI system starts the PILOT interpreter presenting the course Introduction to DROLS Retrieval. Soon you see:

PILOT INTERPRETER VERSION 1.2

and after a pause

INTRODUCTION TO DROLS RETRIEVAL - VERSION 3.4

The version number may change. The course is meant to be self-explanatory.

If the course terminates abnormally, enter the following commands to restart CAI:

```
@END[return]
@CAI[return]
```

5. If the course terminates normally, the message "PILOT CAI PROGRAM DONE" appears and you are disconnected and your carrier light goes out. Turn off your terminal and modem and hang up the telephone.

6. Some CAI user commands are available. These are explained at the beginning of the course. One user command is NOTE. Please use it to comment on the course. If you want to talk baout it, include your name and phone number in a comment.

7. All your responses to course prompts are recorded in a user response file in the DTSS and may be used to assess the course.

8. For assistance call:

Rich Thornett, DTIC-EA, (703) 274-7661 or Autovon 284-7661.

APPENDIX B

HOW TO USE THE DTIC CAI COURSE "INTRODUCTION TO DROLS RETRIEVAL" IN AN IBM PERSONAL COMPUTER

The PC11P PILOT interpreter and course IDR can be put on two double-sided, double-density 5 1/4" floppy diskettes and sent to learners. Disk 1 contains the following files:

Disk	File	Content
1	PILOT.EXE	PC11 PILOT interpreter
1	IDRHELP	Help file for course IDR
1	IDR-AB.BAT	Commands for using A and B drive
1	IDR-AC.BAT	Commands for using A and C drive
1	IDR-C.BAT	Commands for using C drive only
2	IDR	Course IDR

Each of the three BAT file contains just 1 line:

Current Drive	BAT File	Content
A:	IDR-AB.BAT	PILOT B:IDR IDRHELP
A:	IDR-AC.BAT	PILOT C:IDR IDRHELP
C:	IDR-C.BAT	PILOT C:IDR IDRHELP

In the AB and AC cases, if the current drive is not the A drive but, for example the B drive, change to the A drive by entering:

```
B>A:
```

In the AB case, put disk 1 in the A drive and disk 2 in the B drive, and then enter:

```
A>IDR-AB
```

In the AC case, put disk 2 in the A drive and copy the disk 2 files to the C drive, like this:

```
A>COPY A:*.*  C:
```

Then remove disk 2, put disk 1 in the A drive, and enter:

```
A>IDR-AC
```

In the C drive only case, copy the files from both disks to the current directory on the C drive. Do this by putting each disk in the A drive and entering:

```
COPY A:*.*  C:
```

When the files from both disks have been copied, change to the C drive, if necessary, and then enter:

C>IDR-C

When the line in the BAT file is processed, PILOT is copied into the PC's primary memory. If PILOT is too large for it, you will get an error message. Otherwise the PC11P PILOT interpreter starts presenting the course "Introduction to DROLS Retrieval." Soon you will see:

PILOT INTERPRETER VERSION 1.2

and then

INTRODUCTION TO DROLS RETRIEVAL - VERSION 3.4

The version numbers may change. The ciourse is meant to be self-explanatory.

The CAI user commands described in section 5 of this manual are explained at the beginning of the course.

For assistance call:

Rich Thornett, DTIC-EA, (703) 274-7661 or Autovon 284-7661

APPENDIX C

PC11 PILOT ERROR MESSAGES

HSEQRAN ERROR MESSAGES

Each error makes the output file useless unless otherwise stated.

01 Invalid character in term.
Each character in a help glossary term must be a space, a decimal digit, a letter, or one of the following:

```
!"#$%&'()*+,-./:;<=>?@[\]^_
```

Lower case letters are converted to upper case.

02 Term truncated to 60 characters.
A term in the input glossary file contains more than 60 characters. Characters after the 60th are dropped. This message is only a warning. The error will not make the output file unusable.

03 Input term out of sequence.
The input terms must be sorted. The sorting sequence depends on the characters in the terms. These depend on the numeric values of the characters in the ASCII code. The space character is first. The others are ordered as follows:

```
!"#$%&'()*+,-./0..9:;<=>?@A..Z[\]^_
```

where 0..9 and A..Z represent the digits and letters. Lower case is converted to upper before sequence checking.

04 Term directory overflow.
The input file contains more terms than the term directory can hold. Current maximum is 1188. If expansion is needed, contact CAI staff.

05 OUTPUT PILOT HELP FILE UNUSABLE.
The input file contains a fatal error. The output file was made useless.

SEQRAN ERROR MESSAGES

Each error makes the output file useless unless otherwise stated.

01 Too many lines in PILOT program.
Split program into two. Current maximum is 256000 lines.

02 PILOT program line directory overflow.
Each entry in directory corresponds to a program line storage block. Current maximum is 1534 blocks. Split program into two. If directory expansion needed, contact CAI staff.

03 Line label too long.
Line label contains more than 12 characters.

04 Duplicate line label.
The same label is used in two or more lines. Change one of the labels and JUMP or USE commands which refer to the changed label.

05 Unit line label out of sequence.
Each unit line label (four-digit number) must be greater than each unit line label before it. Change the out of sequence label or one or more preceding unit line labels so that each is greater than all those beofre it.

06 Line label table overflow.
The program contains too many line labels (not counting unit labels). The current maximum is 600. If expansion is needed, contact CAI staff.

07 Unit line label table overflow.
The program contains too many unit line labels. The current maximum is 600. If expansion is needed, contact CAI staff.

08 No label or command in line.
Each input line is squeezed and trimmed. If it then contains one or more characters but does not contain a line label or a PILOT command, this error message is emitted. If it contains no characters, it is not an error and is replaced by an empty remark statement (R:). This is done so that each line will have the same number in the input and output files. Squeezing removes characters whose codes are not between 32 (space) and 126 (tilde). Trimming removes spaces at the end of a line (trailing spaces).

09 No colon in command line.
Each input line which contains a command must contain a colon (:) after the command. There may be a parenthetical execution condition between the command name and the colon.

10 No target label in JUMP or USE command.
Each JUMP or USE command must contain an argument specifying the line to be executed next. The argument must be a line label or, in a JUMP command, @A or @M.

11 OUTPUT PILOT PROGRAM FILE UNUSABLE.
The input file contains a fatal error. The output file was made unusable by filling its line directory with zero bits.

PILOT 1100 ERROR MESSAGES

01 Pilot program empty.
No lines in program. Rerun SEQRAN preprocessor with correct course file name.

02 Error in PILOT program line directory.
Rerun SEQRAN preprocessor with correct course file. If error still occurs, contact CAI system staff.

03 Error in reading PILOT program line.
Rerun SEQRAN preprocessor with correct course file. If error still occurs, contact CAI system staff.

04 Label length not 1-12 characters.

05 No valid command name in line.
Each PILOT program line must contain a command name.

06 Error in command execution condition.

07 No colon in line.
Colon (:) must follow PILOT command and its optional parenthetical execution condition expression.

08 Error in ACCEPT command argument.
Argument is optional in ACCEPT command, but if present must be correct.

09 No receiving variable named in COMPUTE command.
COMPUTE command argument starts with a variable type indicator followed by the name of the variable which is to receive a value.

10 No equal sign in COMPUTE command.
COMPUTE command must have equal sign after name of variable to receive a value.

11 No value expression in COMPUTE command.
COMPUTE command must specify a value to be given to the variable. The value is an expression to the right of the equal sign.

12 No target label in JUMP command.
JUMP command argument must specify the line to jump to. This is done by a line label or @A or @M.

13 JUMP target label undefined.
Program line label specified by argument of JUMP command does not exist in the program.

14 JUMP to self not allowed.
JUMP command specifies its own line label as target. This is not allowed, since it can cause an endless loop.

15 No comparand in MATCH command.
MATCH command argument must be a list of strings to be compared with user response.

16 USE command (subroutine call) stack overflow.

Number of subroutine calls (USE commands) minus number of returns from subroutines (END commands) exceeds limit. Current limit is 10.

17 String name length not 1-12 characters.

18 String name character invalid.
Each character in a string variable name must be a letter, a a digit, a hyphen, or underscore.

19 String table overflow.
Too many string variables used in program. Current limit is 24.

The following errors occur in expression evaluation. They are not fatal, that is, the program keeps running. The expression may be due to the program alone or may contain a user response or part(s) of one.

20 Too many left parentheses in expression.

21 Too many right parentheses in expression.

22 Non-digit in fraction in real number in expression.

23 Real number in expression not well-formed.

24 Bad numeric variable name in expression.
Invalid name follows numeric variable type indicator in expression.

25 Expecting operator missing.

26 Unary operator missing where expected.

27 Binary operator missing where expected.

28 NOT operator where binary operator expected.

29 Arithmetic operand with unary Boolean operator.
Operands must be of same type as operator.

30 Boolean operand with unary arithmetic operator.
Operands must be of same type as operator.

31 Arithmetic operand with Boolean operator.
Operands must be of same type as operator.

32 Boolean operand with arithmetic operator.
Operands must be of same type as operator.

33 Boolean operand in arithmetic comparison.
Operands must be of same type as operator.

34 Unidentified operator in polish stack.
Contact CAI staff.

35 Real variable value table overflow.
Contact CAI staff.

36 Expression evaluation stack overflow.
Expression too complex. Current limit is 80.
Try using two or more smaller expressions.

37 Expression evaluation stack underflow.
Contact CAI staff.

38 Expression evaluation parenthesis stack overflow.
Proceding from left to right in expression, the number of open parenthesis characters encountered minus the number of close parenthesis characters met is too large. Current limit is 16.

39 Expression evaluation parenthesis stack underflow.
More close parenthesis characters in expression than open parenthesis characters.

40 Polish stack overflow.
Contact CAI staff. Current limit is 80.

APPENDIX D

TRANSFERRING FILES BETWEEN THE DTSS AND A MICROCOMPUTER

Text files may be sent from the DTSS computer to a microcomputer or from a micro to the DTSS. Here we consider using a microcomputer with a modem and a communications software program, such as CROSSTALK, KERMIT or SMARTCOM.

Such communication programs have a command mode and a connect mode. In COMMAND MODE what you type is taken as a command to the communication program. In CONVERSATION MODE, what you type goes out through your communication port to your modem and through it to a computer or other device if you are connected.

In command mode you can set communication parameters and enter certain of the microcomputer's Disk Operating System (DOS) commands. You can, for example, show a disk's file directory and delete files. Sending in either direction requires a file ready to receive the data, entering a receive command at one end, and entering a send command at the other.

To transmit a file you must start a sending processor at one end and a receiving processor a the other. In the case of Kermit, there is a Kermit processor in the DTSS called KERMIT1100 and a Kermit processor in the microcomputer called MSKERMIT. Each of these processors can send or receive.

To send a file FROM THE DTSS TO A MICROCOMPUTER using Kermit, you first start MSKERMIT be entering:

```
MSK 1200
```

This puts you in command mode. Enter CONNECT to go connect mode, dial the DTSS and sign on. Then assign the file to be sent and start KERMIT1100. For example:

```
@ASG,A COURSE-2.
@ADD A.KERMIT
```

Then return to command mode by entering:

```
Ctrl-]c
```

that is, hold down the Ctrl key, press], release the Ctrl key and then press c. Then enter GET and respond to the prompts as in the following example:

```
Kermit-MS>GET
 Remote source file: COURSE-2.
 Local destination file: COURSE2
```

Note that the file names need not be the same. During the transfer you will see screens like the following:

```
File name: COURSE-2. AS COURSE2
```

```
KBytes transferred: 1

        Receiving: In progress

 Number of packets: 19
 Number of retries: 0
        Last error: None
     Last warning: None
```

When the file has been transferred, Kermit will emit a beep, you will be in command mode, and the screen will look like this:

```
        File name: COURSE-2. AS COURSE2
KBytes transferred: 8

        Receiving: Completed

 Number of packets: 128
 Number of retries: 0
        Last error: None
     Last warning: None
```

Enter:

```
CONNECT
@
```

The first command puts you thorugh to the DTSS. The second entry stops Kermit1100. Sign off the DTSS in the usual way, return to command mode and enter:

```
QUIT
```

which terminates MSKERMIT. This completes the DTSS-to-micro file transmission process.

To send a file FROM THE MICRO TO THE DTSS using Kermit, start MSKERMIT and sign onto the DTSS as explained above. Then establish a reception file in the DTSS as in the following example:

```
@FREE LESSON-7.
@DELETE,C LESSON-7.
@ASG,UP LESSON-7.,F40///1000
```

The FREE and DELETE commands remove file LESSON-7 if one exists.

Now return to command mode as explained above and enter:

```
SEND LESSON7 LESSON-7.
```

During the transfer you will see screens like the following:

```
        File name: LESSON7 AS LESSON-7.
KBytes transferred: 0
Percent transferred: 7%
```

```
                Sending: In progress

         Number of packets: 10
         Number of retries: 0
                Last error: None
            Last warning: None
```

When the file has been transferred, Kermit will emit a beep, you will be in command mode, and the screen will look like this:

```
                File name: LESSON7 AS LESSON-7.
          KBytes transferred: 8
        Percent transferred: 100%

                Sending: Completed

         Number of packets: 128
         Number of retries: 0
                Last error: None
            Last warning: None
```

When the file has been transferred, Kermit will emit a beep and you will be in command mode. Enter:

```
CONNECT
@
@FREE LESSON-7.
```

The first command puts you thorugh to the DTSS. The second entry stops Kermit1100. The FREE command tells EXEC to update the LESSON-7's directory entry to reflect the its current condition. The file may now be examined or modified using the ED processor or presented by PILOT 1100.

APPENDIX E

MAINTENANCE OF PC11T PILOT SYSTEM IN THE DTIC ADPE TIME SHARING SERVICE (DTSS)

This appendix was written under the assumption that the reader will have studied section 2.1 of this manual.

A computer-assisted instruction (CAI) system which presents the course "Introduction to DROLS Retrieval" is in the DTIC ADPE Time Sharing System (DTSS). This system is hosted by the DTIC Univac 1100/61 computer. Instructions for learners using this system are given in Appendix D of this manual.

This appendix describes the components of the PC11T PILOT presentation system and how it works. It discusses restoring the system in case files are lost or damaged and updating the course "Introduction to DROLS Retrieval" (IDR).

1. In the DTSS actions take place in the following projects and accounts:

PROJECT	ACCOUNT	USE
CAI	DTDDDCAI0001	Prepare CAI
CAITEACH	DTDDDCAI0002	Present CAI
CAILEARN	DTDDDCAI0003	Receive CAI

We will discuss files and processors in each of these groups. But first let's review some fundamentals about files and elements.

PROGRAM FILES are files which contain separately accessible parts which may be referred to by name. These parts are called elements. There are three kinds of elements. SYMBOLIC elements contain text. RELOCATABLE elements are produced by compilers and assemblers. ABSOLUTE elements are executable programs or processors. An element specifier consists of a file name followed by a period followed by an element name. The rules for element names are the same as those for file names.

Files which are not program files can contain many different kinds of data. A text file contains a sequence of lines of text. A text file can be copied into a symbolic element of a program file, and the other way around. For example, to copy file PP1 into A.PP1 enter:

```
@ED,IQ A.PP1
@EDIT
ADD PP1.
EXIT
@FREE A.
```

To copy element A.PH1 to file PH1 enter:

```
@FREE PH1.
@DELETE,C PH1.
@ASG,U PH1.,F33///1000
@ED,IQ PH1.
```

```
@EDIT
ADD A.PH1
EXIT
@FREE PH1.
```

An "ADD" file or element is one which contains ECL commands and, where appropriate, related data. For example, if file XXX contains one of the last two examples and element A.YYY contains the other, then entering:

```
@ADD XXX
@ADD A.YYY
```

will accomplish both copies.

2. CAI PREPARATION PROJECT FILES AND PROCESSORS

The CAI preparation project (CAI) contains files and processors used in preparing and testing PC11T PILOT programs, courses and help files. The following files are especially important: A, ED$PF, MACBACKUP, PILOT, IDR0013, IDRHELP and JL. (We will sometimes omit the file name qualifier and asterisk.)

File CAI*A is an all-purpose program file. Element A.U is "added" each time someone starts a run specifying project CAI. This sets the backspace to the backspace character and makes text editor (ED) macros available.

File CAI*ED$PF (synonym E) is a program file containing text editor macros as symbolic elements. The text editor (ED) can be used to put new macros in this file or to modify existing macros.

File CAI*MACBACKUP (synonym: ED$PF) contains macros in executable form. This file has a special structure.

Program file CAI*PILOT contains the following files:

Element	Type	Use
TIDY	Processor	Improves course file appearance
SEQRAN	Processor	Preprocesses a raw PC11T PILOT program
HSEQRAN	Processor	Preprocesses a raw PC11T PILOT help file
DATA	Processor	Presents a PC11T PILOT program
SEQRANX1	Add	Applies SEQRAN to PP1 producing PP1R
HSEQRANX1	Add	Applies HSEQRAN to PH1 producing PH1R
X1	Add	Presents course PP1R using help file PH1R

Calling the interpreter DATA makes it possible for the learner to input lines starting with the character @.

File CAI*IDR0013 contains course IDR in raw form.

File CAI*IDRHELP is contains in raw form the help file used with course IDR.

File JL contains an element for each lesson in course IDR. These are named JL.00, JL.01 through JL.13.

3. CAI PRESENTATION PROJECT FILES AND PROCESSORS

The CAI presentation project (CAITEACH) contains files and processors used in presenting PC11T PILOT courses. The following files are especially important: PILOT, PILOTPROGRAM, PILOTHELP and URFP.

File LEARN*PILOTPROGRAM contains in presentable form the current version of PC11T PILOT course IDR.

File LEARN*PILOTHELP contains in presentable form the current version of the help file used with course IDR.

File LEARN*URFP contains the four-character suffix of the user response file most recently created. For example, if URFDE67 was the last userresponse file, URFP contains DE67.

File LEARN*PILOT is a program file containing the following elements:

Element	Type	Use
MENU	Add	Calls processors A, B and DATA
A	Processor	Assigns file URFP
B	Processor	Updates file URFP, assigns course, help and user response files
DATA	Processor	Presents PILOTPROGRAM using file PILOTHELP
END	Add	Restores ECL mode
ENDOFF	Add	Restores ECL mode and disconnects user

When EXEC encounters the command @CAI in a user runstream, system processor SYS$*LIB$.CAI is started. It gives EXEC the command @ADD LEARN*PILOT.MENU. This causes the three commands in element MENU to done in sequence:

```
@LEARN*PILOT.A
@LEARN*PILOT.B
@LEARN*PILOT.DATA
```

Processors A, B and DATA are Pascal programs. A and B are short. They are needed because a Pascal 1100 processor cannot read or write a file which is not assigned to the run when the processor starts.

Processor A merely assigns file URFP.

Processor B reads the four-character suffix from URFP, computes the next suffix, writes it back to URFP, combines it with the string URF to form a 7-letter user response file name, like URFBC45. B then tells EXEC to create the user response file, catalog it, and make the name USERRESPONSE a synonym for it. Finally B tells EXEC to assign files PILOTPROGRAM, PILOTHELP and USERRESPONSE.

Processor DATA, the PC11T PILOT interpreter, is started by the third command in PILOT.MENU and presents PILOTPROGRAM, which is course IDR. The interpreter is called DATA because starting a processor called DATA changes the run's mode from control mode to data mode, where user responses which begin with the character @ are not treated as ECL commands.

When the learner enters the user command STOP or reaches the end of the course, the interpreter gives EXEC one of two commands and terminates. The two commands are: @ADD LEARN*PILOT.END and @ADD LEARN*PILOT.ENDOFF. Element END contains one command, @END, which puts the run in control mode so that subsequently entered ECL commands will be recognized. Element ENDOFF contains two commands: @END and @XQT LEARN*PILOT.SIGNOFF. The XQT command tells EXEC to execute program SIGNOFF, which disconnects the user.

If another course is added to the curriculum, PILOT.MENU will be modified so that the user selects a course from a menu, and the interpreter presents that course.

The processors named in this appendix correspond to Pascal source elements in file CAI*RGTC as follows:

Pascal Source	Processor
CAI*RGTC.1T	CAI*PILOT.TIDY
CAI*RGTC.1A	CAI*PILOT.SEQRAN
CAI*RGTC.1B	CAI*PILOT.HSEQRAN
CAI*RGTC.2A	LEARN*PILOT.A
CAI*RGTC.2B	LEARN*PILOT.B
CAI*RGTC.2H	LEARN*PILOT.DATA

4. RESTORING AND UPDATING DTSS CAI SYSTEM COMPONENTS

Backup copies of DTSS files are made frequently by DTIC-ZDT. If a file which is part of the PC11T PILOT system is damaged or disappears, call 274-6855 and request restoration of the file.

To update course IDR, use the ED processor. If the modification is sizeable, make a copy of the part to be modified, such as a course unit or section, and modify and test it separately. Then use ED to delete the current version of the part and insert the new version.

To extract a part, make a copy of IDRE and delete the lines before and after the ones to be modified. The ED command split can also be used. To replace a part, delete the old part and copy in the new part. The ED commands delete and add are use to delete lines and copy in the contents of a file or element. For example:

```
@ED,U  IDRE.
D  100  200
ADD+  NEWPART.
EXIT
@FREE IDRE.
```

These commands replace lines 100 to 200 of file IDRE with the contents of file NEWPART.

APPENDIX F

MAINTENANCE OF PC11P PILOT CAI SYSTEM

The PC11P PILOT system is a computer-assisted instruction (CAI) system which presents courses using IBM Personal Computers (PCs) or compatibles. This system consists of hardware, software and files. The hardware is a PC with keyboard, monitor and at least two floppy diskettes or one fixed disk. The software consists of processors used to prepare and present courses. The preparation processors include a text editor or word processor and some special processors. The presentation processor is the PC11P PILOT interpreter. The files contain processors, PC11 PILOT courses and help files, and command sequences (BAT files). The preparation and presentation files and processes are described in section 2 of this manual.

The first course developed for this system is "Introduction to DROLS Retrieval" (IDR).

This appendix describes the files and processors in the PC11P PILOT system. It discusses putting the system on diskettes and using these, saving the system's files and restore lost or damaged files, and modifying course IDR.

1. COURSE PREPARATION AND PRESENTATION FILES AND PROCESSORS.

The microcomputer used to prepare course IDR is an IBM PC which has two floppy disk drives and a 10-megabyte fixed disk drive. These are drives A, B and C, respectively. The A drive is used for system startup and for communicating with the DTSS using the MSKERMIT communication program (see Appendix C). Three directories in the C drive are used: the main directory, the CAI directory and the PIL directory.

The ROOT directory contains common commands, such as EDLIN, PRINT, BACKUP and RESTORE.

The CAI directory contains each of the 14 lessons in the course in its own file in editable form. These files are named JL00.NNN through JL13.NNN, where the extension NNN is the version number of the lesson. To combine the fourteen lessons into course IDR in file IDRE, enter:

```
COPY  JL00.100+JL01.100+JL02.100+JL03.100+JL04.100+
      JL05.100+JL06.100+JL07.100+JL08.100+JL09.100+
      JL10.100+JL11.100+JL12.100+JL13.100  IDR
```

all in one line. File MI0013.BAT contains this command.

To modify file IDRE, use EDLIN or another processor to modify one or more lesson files. Apply the TIDY processor to the lesson, if appropriate, like this:

```
TIDY  JL01.005  TY
COPY  TY  JL01.005
ERASE TY
```

The same result can be obtained by entering:

```
T  JL01.005
```

When the lessons have been modified, use MI0013.BAT to remake file IDRE. If you change a lesson name, change it in the BAT file too. File IDRHELPE contains in editable form the help file used with course IDR.

The PIL directory contains the Pascal source and object files of the components of the TIDY, SEQRAN and HSEQRAN preprocessors and the PC11P PILOT interpreter, and the BAT files and related input files used to compile and link these components. These are listed in section 4 below. The following processors and BAT files are relevant here:

File	Use
TIDY.EXE	Executable TIDY processor
MI0013.BAT	Command which makes file IDRE from lesson files JL00.nnn through JL13.nnn
SEQRAN.EXE	Executable SEQRAN processor
SPP1.BAT	Commands which tell processor SEQRAN to convert editable course PP1 to presentable course PP1R
HSEQRAN.EXE	Executable HSEQRAN processor
PILOT.EXE	Executable PILOT processor
XPP1.BAT	Commands which start presentation of course PP1R
IDR-AB.BAT	Start A and B drive system presenting IDR
IDR-AC.BAT	Start A and C drive system presenting IDR
IDR-C.BAT	Start C drive only system presenting IDR

To prepare the IDR course and help files for presentation, first get into the PIL directory and copy them into it from the CAI directory:

```
CHDIR  \PIL
COPY  \CAI\IDRE
COPY  \CAI\IDRHELPE
```

To convert file IDRE to presentable form enter:

```
SEQRAN  IDRE  IDR
```

To convert file IDRHELPE to presentable form enter:

```
HSEQRAN  IDRHELPE  IDRHELP
```

To present course IDR using help file IDRHELP enter:

```
PILOT  IDR  IDRHELP
```

2. PRESENTATION DISKETTES SENT TO LEARNERS

The PC11P PILOT interpreter and course IDR can be put on two double-sided, double-density floppy diskettes and sent to learners. Disk 1 contains the following files:

Disk	File	Content

1	PILOT.EXE	PC11 PILOT interpreter
1	IDRHELP	Help file for course IDR
1	IDR-AB.BAT	Commands for using A and B drive
1	IDR-AC.BAT	Commands for using A and C drive
1	IDR-C.BAT	Commands for using C drive only
2	IDR	Course IDR

Each BAT file contains just 1 line:

File	Content
IDR-AB-BAT	PILOT B:IDR IDRHELP
IDR-AC.BAT	PILOT C:IDR IDRHELP
IDR-C.BAT	PILOT IDR IDRHELP

In the AB and AC cases, put disk 1 in the A drive and disk 2 in the B drive, and then enter:

```
A:              or        A:
IDR-AB                    IDR-AC
```

In the third case, the contents of both disks must be copied to the current directory on the C drive. Put each disk in the A drive and enter:

```
C:
COPY A:*.*
```

Then enter:

```
C:
IDR-C
```

When the line in the BAT file is processed, PILOT is copied into the PC's primary memory. If PILOT is too large for it, you will get an error message.

To prepare a pair of diskettes, use the DOS FORMAT command to format them, and copy the files from the C drive to them. Or use the DOS DISKCOPY command to copy from prepared diskettes to other diskettes.

3. SAVING AND RESTORING FILES

To save and restore the files which constitute the PC11P PILOT system, use the DOS BACKUP and RESTORE commands. Only C drive directories CAI and PIL need be saved.

4. RESTORING AND UPDATING DTSS CAI SYSTEM COMPONENTS

To update course IDR, use the EDLIN processor. If the modification is sizeable, make a copy of the part to be modified, such as a course unit or section, and modify and test it separately. Then use EDLIN to delete the current version of the part and insert the new version.

To extract a part, make a copy of IDRE and delete the lines before and after the ones to be modified. To replace a part, delete the old part and

copy in the new part. The ED commands delete and transfer are use to delete lines and copy in the contents of a file or element. For example:

```
@EDLIN  IDRE
100  200  D
TNEWPART
E
```

These commands replace lines 100 to 200 of file IDRE with the contents of file NEWPART.

5. PASCAL-RELATED FILES AND THEIR USE

The following Pascal-related files are in directory PIL:

File	Use
TIDY.PAS	TIDY processor source
TIDY.EXE	Executable TIDY processor
SEQRAN.PAS	SEQRAN processor source
LSEQRIP	Input to LSEQRAN.BAT
LSEQRAN.BAT	Commands which link SEQRAN processor
SEQRAN.EXE	Executable SEQRAN processor
HSEQRAN.PAS	HSEQRAN processor source
LHSEQRIP	Input to LHSEQRAN.BAT
LHSEQRAN.BAT	Commands which link HSEQRAN processor
HSEQRAN.EXE	Executable HSEQRAN processor
PIMAIN.PAS	Main part of PC11P PILOT interpreter source
PIXOP.PAS	PILOT command processor source
PIXPEV.PAS	PILOT expression evaluator source
PIHELP.PAS	PILOT help command processor source
HEXDUMP.PAS	Hexadecimal dump subroutine source (for testing)
LPILOTIP	Input to LPILOT.BAT
LPILOT.BAT	Commands which link PILOT processor
PILOT.EXE	Executable PILOT processor

This list does not include the object file (extension OBJ) corresponding to each source file (extension PAS) and having the same filename (part before dot).

To produce an object file form a source file requires two passes (steps). For example, to produce PIXOP.OBJ from PIXOP.PAS enter:

```
A:PAS1  PIXOP PIXOP NUL NUL
A:PAS2
```

To link PILOT's object files and produce an exectuable file enter:

```
LPILOT
```

These commands presuppose that the A drive holds a diskette containing the following IBM files: ENTX6S, FILKQQ.INC, FILUQQ.INC, LINK.EXE, PAS1.EXE, PAS2.EXE, PASCAL, PASCAL.LIB and PASKEY.

An explanation of compiling and linking IBM Pascal programs is beyond the

scope of this manual.

END

DTIC

6-86