SECTION 4

PROGRAM DEBUGGING

Chapter 3 described the basic facilities of Instant Pascal for creating, changing, and executing a Pascal program. In addition, the Step (+<,>) command and DEL key have been shown to be useful for analyzing the dynamic behavior of a program. This chapter describes several additional facilities which are available for program debugging.

4.1 TRACING

When the Step command is used, tracing is automatic. In general there are two kinds of tracing, which can be independently controlled in Instant Pascal during continuous execution (+<G>).

a. Statement tracing

b. Assignment tracing

These two kinds of trace may be independently controlled by two toggles, called S and A, respectively. These toggles are switched by the S and A commands.

4.1.1 Statement Tracing (S Command)

Statement tracing prints each text unit just before it is executed.

Detail:

The state of the toggle after entering the S command is indicated by the word ON or OFF printed immediately after the command.

4.1.2 Assignment Tracing (A Command)

Assignment tracing prints values changed by assignment and FOR statements.

Details:

a. The state of the toggle entering the A command is indicated by the word ON or OFF printed immediately after the command.

b. If S is off and A is on, assignment statements are nevertheless printed in order to associate them with the printed values. FOR statements are also printed, but only the first time through the loop.

4.2 IMMEDIATE STATEMENT EXECUTION (X COMMAND)

It is possible, under well defined conditions, to key in a single statement and have it executed immediately, independent of the execution of the program in memory. This is useful during a break or after an abnormal program termination for examining and changing the values of variables.

The +<X> (eXecute) command immediately sets up a text input prompt. Key in one statement; it will be executed upon depression of RETURN. Note the restrictions to the use of X which are listed below.

Details:

a. There must be a program in memory which has been successfully bound and whose execution has begun. That is, X should be executed only during a break or after completion of a program, either a normal completion or one accompanied by an ERROR message. Other use of X will lead to unpredictable results.

b. If the program is not bound when X is executed, it will first be bound before the statement is executed. This program binding is indicated by the printout ".C.". If it leads to a diagnostic, the statement will not be executed.

c. The statement must itself be bound before it can be executed. This can lead to diagnostics, in which case the statement will not be executed.

d. There is the question of visibility of identifiers. That is, the statement WRITELN(A[I]) may have different interpretations of A and I depending on the block in which it is presumed to be executed. This block is identified by the position of the text unit pointer at the time +<X> is executed. The text unit pointer must be in the statement part of an active block or at the bottom of the program. The movement commands used in text editing (see 3.2, 3.4) may be used to position the pointer before using X.

e. Note that at a break, the text unit pointer is at the text unit listed and about to be executed.

f. If the program terminates with an ERROR message, the text unit pointer is at the text unit listed, i.e., the one whose execution caused the error.

g. If the program is bound in response to X (i.e., if ".C." is printed), and there are no Binder diagnostics, the interpretation of the statement will be with respect to the statement part of the main program, since, in this case, the Binder leaves the text unit pointer at the bottom of the program.

h. The single statement executed can be a structured statement. Its complexity, therefore, is limited only by the 60-character length of the AIM 65 input line.

SECTION 5

INSTANT PASCAL TEXT UNITS

The internal form of a Pascal program in the Instant Pascal system has been designed to satisfy three objectives not previously considered compatible in systems supporting structured languages:

- The internal form of the program must be backwards translatable to equivalent source code. This objective permits building a system which can support the illusion of directly executing the source code.

- Execution time should be insensitive to complexity of data or statement structures. For example, the time required by an IF statement to skip over an unexecuted compound statement should be about the same whether the compound statement is three lines or one hundred lines long.

- The consequences for the structure of the internal form of the program of any source-program change must be confined to the portions of the program which are changed. As a counterexample, changing one letter, VAR A,B:SEAL; to VAR A,B:SEAT; (where SEAL and SEAT are defined types) in a compiler-based system can completely alter the object program and therefore requires total recompilation.

The approach which has been taken to reconcile these objectives in Instant Pascal divides the process of transforming source text into answers into three phases.

a. Translation (elicited by the R, I, and X commands) processes Pascal source code and translates it into the internal program format which is in executable form except for certain addresses and integer values, such as pointers to variable declarations and lengths of structured types. The proper values cannot be computed for these values because the program cannot be assumed to be whole. Space is left, however, for these values to be later added. (These entities are called "spanners" in this section).

b. Binding (elicited by the C and X, and possibly by the G and step commands) scans the entire program (or in the case of X, the statement), verifying its structural integrity and assigning values to all spanners.

c. Execution (elicited by the G, step, and X commands) sequentially interprets program statements and alters data values in accordance with the rules of Pascal.

5.1 TEXT UNITS IN EDITING

In order to ensure that the user's freedom to edit programs is not seriously constrained by the design approach taken above, this design approach also incorporates the following principle.

> Every program is conceived as being built of a single-level, linear sequence of syntactic building blocks called "text units". From the editor's point of view, the text unit is the throwaway unit. Program building and altering reduces to inserting text units into, and deleting text units from, this linear sequence.

As a consequence of this principle, character-level editing within a given text unit is accomplished by deleting the text unit and inserting a replacement. In some cases, text units are small enough that the editing process, viewed as deletion followed by insertion, is essentially the same as conventional character-oriented editing. For example, the following are text units:

BEGIN, END, ; (as a statement separator), REPEAT, ELSE

On the other hand, some text units are larger. For example, all simple statements are text units; in fact, expressions are always contained within larger text units.

The person familiar with Pascal will readily incorporate the text-unit orientation into his or her editing style. Here is a brief and informal definition of most of the Pascal text units.

a. Program, procedure, and function headings, including formal parameter lists.

Example:

```
FUNCTION ISPRIME(N:2..MAX):BOOLEAN;
```

b. Certain part header words:

```
CONST
TYPE
VAR
```

c. Constant definitions, including the final semicolon.

Example:

```
GREETING='HELLO,THERE!';
```

d. Type definitions, including the final semicolon, except in the case of record type definitions, in which case the word RECORD stands in place of the type and final semicolon.

Examples:

```
TABLE=ARRAY[RANGE]OF COST;
HASHTABLE=RECORD
```

e. Variable and field declarations (they are the same).

Example:

```
GAP=$A409,INPUTCHAR:CHAR;
```

(Note that the final semicolon after a field declaration is always required in Instant Pascal unless the field type is RECORD.)

f. END and ; at the end of a RECORD definition.

g. Simple statements.

Examples:

```
GOTO 5
SUBR(CRLOW)
A:=B[N+1]*C
```


h. The connective fragments which are used to build structured statements.

Examples:

```
REPEAT
UNTIL A=N
IF ODD(I) THEN
WITH R[P] DO
```

i. Labels and case constant lists preceding statements.

j. Comments.

Example:

```
(*THIS IS A COMMENT*)
```

In general, the lister begins each text unit on a new line. Exceptions occur when labels and case constants precede statements and, most commonly, when semicolons follow statements. This fact is important in editing because the formatted listing is frequently used as a reference document in the editing process, and the editing commands which require numeric parameters (+<L>/n, +<K>/n, +<U>/n, +<D>/n) always count text units, not lines. The most common case to remember when counting text units on a listing is to count as two text units any line containing a simple statement or a structured statement connective fragment followed by ";".

Examples:

```
A:=1;
END;
UNTIL NOT ODD(N);
```

There is a special case in the input translator which is necessitated by a syntactic ambiguity in Pascal. Consider the plight of the input translator on encountering the following text in the course of program creation.

```
A,B,C,D:E;
```

Is it a variable declaration, or is it a case constant list followed by a procedure statement? These two objects must be translated differently. This is the only case in which the translation process is context-sensitive. The translator decides how to treat this input by examining a context flag of type (DECLARATIVE,STATEMENT). The flag's value is maintained according to the following rules.

a. It is initialized to DECLARATIVE.

b. Otherwise, its value depends on the identity of the text unit immediately preceding the text unit pointer, and the value is subject to change every time the identity of the preceding text unit is changed, according to the following rules:

(1) If there is no preceding text unit, the value is unchanged.

(2) If the preceding text unit is END or the separate ";", the value is unchanged.

(3) Otherwise, if the preceding text unit is one which occurs in declaration or definition parts, the value is changed to DECLARATIVE; or if the preceding text unit is one which occurs in the statement part, the value is changed to STATEMENT.

The reason for excepting END and the separate semicolon is that they can occur in both statements and at the end of RECORD types.

5.2 DEFINITIONS OF TEXT UNITS

This section presents a list of the text units of Instant Pascal. It is not a formal definition of the syntax of the source language. Such a definition is implied in Chapter 7. Rather, this section uses the reader's assumed knowledge of existing syntax definitions of Pascal to present the source language from the linear, single-level text unit point of view.

There are two metalinguistic elements used in these lists.

a. Square brackets [ ] enclose an optional element.

b. Elipses ... mean that the previous element is optionally repeated any number of times.

The detail notes accompanying the lists clarify specific points and cite deviations from standard Pascal practice.

5.2.1 Program Heading

```
PROGRAM identifier;
```

5.2.2 Text Units Appearing in Both Declarative and Statement Parts

1. (* comment text *)
2. END
3. ;

Details:

a. Comment text may not contain the character sequence "*)".

b. The character ";" appears at the end of certain heading, declaration, and definition text units. This use of ";" is not a separate text unit but is an inseparable part of the heading, declaration, or definition text unit.

5.2.3 Text Units Appearing in Declarative Parts

The text units are:

1. LABEL integer [,integer]...;
2. CONST
3. identifier=constant;
4. TYPE
5. identifier=type;
6. identifier=RECORD
7. VAR
8. identifier[=$hhhh][,identifier[=$hhhh]]...:type;
9. identifier[=$hhhh][,identifier[=$hhhh]]...:RECORD
10. PROCEDURE identifier[(formal parmeter list)];
11. FUNCTION identifier[(formal parameter list)]:type;

Details:

a. The constant definition identifier=identifier; is not in the language.

b. The option [=$hhhh] denotes a global variable assigned to the absolute hexadecimal address hhhh. If the type is longer than one byte, hhhh is the lowest-numbered address in the variable.

c. The identifier[=$hhhh] cases above are used for both variable and field declarations. The option [=$hhhh] is not permitted in field declarations.

d. The semicolons at the end of cases 1,3,5,8,10,11 are not optional. Therefore, the optional semicolon permitted by Pascal after the last field declaration in a record definition is not optional in Instant Pascal; it is required.

5.2.4 Simple Statements

The statements are:

1. variable:=expression
2. function designator:=expression
3. GOTO integer
4. identifier([actual parameter list)]

Details:

a. Statement type 2 expresses assignment to the value of a function.

b. The integer in the GOTO statement must appear in the LABEL declaration of the block containing this statement part. GOTO may not jump outside its own block.

c. The identifier in the statement type 4 is a declared or predefined procedure name.

5.2.5 Statement Connective Fragments

The connective fragments are:

1. BEGIN
2. END (the same as in 5.2.2)
3. ; (the same as in 5.2.2)
4. IF Boolean expression THEN
5. ELSE

6. WHILE Boolean expression DO
7. REPEAT
8. UNTIL Boolean expression
9. FOR identifier:=expression TO expression DO
10. FOR identifier:=expression DOWNTO expression DO
11. CASE expression OF
12. case constant[,case constant]...:
13. OTHERWISE:
14. WITH record variable[,record variable]...DO
15. label:

Details:

a. Case constants are not strongly type checked against the expression in the CASE fragment. A case constant will be selected at run-time if its ORDinal value matches the ORDinal value of the expression.

b. OTHERWISE: is the default case "constant".

SECTION 6

DIAGNOSTIC MESSAGES

This chapter describes diagnostic messages which can occur during translation, binding, execution, and listing.

6.1 SOURCE INPUT DIAGNOSTICS

Each correct source input line is a sequence of one or more syntactically correct text units. There is syntax checking during translation, but only within each text unit. With the exception noted in 5.1, there is no concern at translation time about the order in which text units are entered or about the overall structure of the program.

During source input (I, R, or X command), an entire line is input before any of the line is translated. This permits free use of the DEL key to correct keying errors. Once translation begins (immediately after keying RETURN) each text unit is translated without regard for any text units either preceding or following it on the same line. If an error is discovered during the translation of any text unit in an input line, no text unit in that line is incorporated into the program. If no error is discovered, all text units in the line are incorporated into the program.

Three diagnostic messages can occur during source input.

- The syntax error message

  #...#

  See 3.1 for a discussion of this message.