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.

o The message

"NO SPACE LEFT"

This indicates that the current number of free memory bytes (see +<M>, 3.3.1) minus the number of bytes into which this line has been translated is negative.

o The message

"TOO MUCH CODE"

This message, which is expected never to occur, says that the 256-byte buffer which receives the output of the translator is not large enough to hold the internal form of the entire input line. If the message occurs and if the input line contains multiple text units, break the input into more than one line.

Details:

a. In the case of the syntax error message, the right-most "#" usually gives a strong clue about the error, since the character immediately to its right is the one which led the syntax analyzer into the impasse. If the right-hand "#" is the last character of the diagnostic message, something is missing at the end of the input, for example, a final semicolon in a declaration or definition.

b. The usual response to the "NO SPACE LEFT" message is to save the program on tape and then to figure out how to do what needs to be done in less space. If a larger memory size could have been declared because RAM is available which wasn't used, it is still necessary to write the program out and then read it back in.

6.2 BINDER DIAGNOSTICS

Diagnostic messages which occur during binding have the following distinctive form: an indicator line describing the error, followed by one or more program text lines describing where the error occurred. The following indicator lines can occur:

SEQUENCE
UNDEF identifier
DUP identifier
UNDECL label
DUP label
TYPE identifier

6.2.1 *SEQUENCE* Diagnostic

The *SEQUENCE* diagnostic indicates a syntax error. This is usually an inter-text-unit syntax error, for example, a missing semicolon, but intra-expression syntax errors which escape the checking in the translator are possible, as the example in 3.4.8 illustrates.

The indicator line is followed by four text units to display the context of the error. The actual error was discovered, that is, the impasse occurred, at or in the first text unit listed.

A *SEQUENCE* diagnostic immediately aborts binding with the text unit pointer at the first of the four listed text units.

Details:

a. If the only thing following the *SEQUENCE* line is ".", this indicates a missing END in the program. Listing the whole program will show this in the form of a nonzero indentation on the last source line.

b. If examination of the program shows that the syntax error is due to something missing immediately before the first listed text unit (a missing semicolon is a common example), it can be immediately entered with the +<I> command without moving the text unit pointer.

6.2.2 Identifier Lookup Diagnostics

The indicator lines *DUP* identifier and *UNDEF* identifier are each followed by one text unit which contains the cited identifier occurrence.

The *DUP* diagnostic is generated at a defining occurrence of an identifier if the identifier has a prior defining occurrence at the same block level.

The *UNDEF* diagnostic is generated at a referring occurrence of an identifier if the identifier has no prior defining occurrence at the same or at any higher block level which would be visible to this referring occurrence.

Neither of these diagnostics stops the Binder.

6.2.3 Label Lookup Diagnostics

The indicator lines *DUP* label and *UNDECL* label are each followed by one text unit which contains the cited label occurrence.

The *DUP* diagnostic is generated at a label: preceding a statement if there was a prior occurrence of the same label: in the statement part of the same block.

The *UNDECL* diagnostic is generated at a GOTO label if there is no label declaration in this block which names the label.

The case in which a label is declared and occurs in a GOTO but does not occur preceding a statement is caught at run-time when (and if) the GOTO is executed.

Neither of these diagnostics stops the Binder. Note that labels are stored and searched as character strings, except that insignificant zeros are suppressed. Instant Pascal does not impose the conventional limit of four digits to the length of a label.

6.2.4 Type Consistency Diagnostics

The indicator line *TYPE* identifier can indicate an inconsistency between the types of the upper and lower bounds of a subrange type; it can also be generated by a STRING type with a noninteger length. The identifier in the indicator line is a type identifier appearing in either the subrange or string type; this identifer refers to a type which is inconsistent with the definition in which it occurs. This definition occurs in the text unit following the indicator line.

This diagnostic does not stop the Binder.

Details:

a. If no Binder diagnostic occurs, the Binder will output the message "OK" and the text unit pointer will be at the top of the program.

b. If a Binder diagnostic occurs but no *SEQUENCE* diagnostic occurs, the text unit pointer will be at the bottom of the program.

c. It is not possible to execute a program without an error-free binding.

6.3 EXECUTION-TIME DIAGNOSTICS

The are 53 possible execution-time diagnostics enumerated in Appendix E. This section deals with what is common to all of them.

All execution-time diagnostics are fatal to execution. The error reported by the diagnostic message is discovered in the process of interpreting a text unit. This text unit is printed as part of the diagnostic message; then execution terminates (with Break-in-progress false) and control returns to the command interpreter with the text unit pointer at the offending text unit.

The form of the diagnostic message is.

```
ERROR #nn
text unit
```

The definitions of error codes nn are listed in Appendix E.

After control returns to the command interpreter, a series of WRITELNs may be executed with the +<X> command to examine the state of the data. The position of the text unit pointer insures that visibility of identifiers is the same as that which prevailed at the time of the error. The data structures built during execution are not initialized until the next +<G> or +<,> is executed, so data are available for examination. (This is true after an error-free termination also, but first do a +<B> before the +<X> command to position the text unit pointer to the bottom of the program).

6.4 SOURCE OUTPUT DIAGNOSTICS

Under normal conditions, the lister will never encounter something which it cannot translate. If memory becomes corrupted,

however, the lister may encounter a value which does not translate meaningfully; this will lead to the output of a single "?" as part of the output text.

Certain corrupted memory configurations can cause the lister to loop continously; it may be necessary to RESET the AIM 65 under these conditions.

Detail:

> Under certain conditions the lister will only print a partial text unit as part of a diagnostic message. This can occur where lists are involved and an error was caught in the middle of a list: in LABEL and VAR declarations, parameter lists, and case constant lists. In one case, namely parameter lists, the backward translation algorithm may become confused if the translation starts in the middle of the list, and an extra semicolon might come out. Also, the text unit pointer may actually be positioned in the middle of a list. This will not lead to difficulty, however, and the text unit pointer can be repositioned as usual. Care should be taken not to delete anything until the text unit pointer is back to the beginning of a text unit.