If the program terminates incorrectly (e.g. without 'END.') then the message 'No more text' will be displayed and control returned to the editor.

If the compiler runs out of table space then the message 'No Table Space' will be displayed and control returned to the editor. Normally the programmer will then save the program on tape, re-load the compiler and specify a larger 'Table size' (see Section 0.0).

If the compilation terminates correctly but contained errors then the number of errors detected will be displayed and the object code deleted. If the compilation is successful then the message 'Run?' will be displayed; if you desire an immediate run of the program then respond with 'Y', otherwise control will be returned to the editor.

During a run of the object code various runtime error messages may be generated (see Appendix 1). You may suspend a run by using CS; subsequently use CC to abort the run or any other key to resume the run.

0.3 Strong TYPEing.

Different languages have different ways of ensuring that the user does not use an element of data in a manner which is inconsistent with its definition.

At one end of the scale there is machine code where no checks whatever are made on the TYPE of variable being referenced. Next we have a language like the Byte 'Tiny Pascal' in which character, integer and Boolean data may be freely mixed without generating errors. Further up the scale comes BASIC which distinguishes between numbers and strings and, sometimes, between integers and reals (perhaps using the '%' sign to denote integers). Then comes Pascal which goes as far as allowing distinct user-enumerated types. At the top of the scale (at present) is a language like ADA in which one can define different, incompatible numeric types.

There are basically two approaches used by Pascal implementations to strength of typing; structural equivalence or name equivalence. Hisoft Pascal 4 uses name equivalence for RECORDs and ARRAYs. The consequences of this are clarified in Section 1 - let it suffice to give an example here; say two variables are defined as follows:

```
VAR A : ARRAY['A'..'C'] OF INTEGER;
    B : ARRAY['A'..'C'] OF INTEGER;
```

then one might be tempted to think that one could write A:=B; but this would generate an error (*ERROR* 10) under Hisoft Pascal 4 since two separate 'TYPE records' have been created by the above definitions. In other words, the user has not taken the decision that A and B should represent the same type of data. She/He could do this by:

```
VAR A,B : ARRAY['A'..'C'] OF INTEGER;
```

and now the user can freely assign A to B and vice versa since only one 'TYPE record' has been created.

Although on the surface this name equivalence approach may seem a little complicated, in general it leads to fewer programming errors since it requires more initial thought from the programmer.