```
    2. INPUT CHOICE
    3. Call CHOICE subroutine
END
```

III 3 (CHOOSE)1 (PRINT menu)

```
BEGIN
    1. Clear screen
    2. PRINT menu and prompt
END
```

III 3 (CHOOSE) 2 (INPUT CHOICE)

```
BEGIN
    1. INPUT CHOICE
    2. Check that CHOICE is within range
END
```

III 3 (CHOOSE) 3 (call CHOICE)

```
BEGIN
    1. CASE OF CHOICE
        ENDCASE
END
```

III-3-1 (PRINT menu) can now be coded into BASIC:

IV 3 (CHOOSE) 1 (PRINT menu) BASIC CODE

```
REM CLEAR SCREEN
PRINT CHR$(12): REM OR 'CLS'
PRINT
PRINT
PRINT
PRINT
PRINT "1. FIND RECORD (FROM NAME)"
PRINT "2. FIND RECORD (FROM INCOMPLETE
    NAME)"
PRINT "3. FIND RECORD (FROM TOWN)"
PRINT "4. FIND RECORD (FROM INITIALS)"
PRINT "5. LIST ALL RECORDS"
PRINT "6. ADD NEW RECORD"
PRINT "7. CHANGE RECORD"
PRINT "8. DELETE RECORD"
PRINT "9. EXIT & SAVE"
```

III-3-2 (INPUT CHOICE) and **III-3-3 (call CHOICE)**, however, need further refinement. Let's look first at the next level of development of **III-3-2**.

Assigning a numeric value to the variable CHOICE is perfectly simple: after the prompt, an INPUT CHOICE command will do this. However, there are only nine possible choices. What would happen if we mistakenly entered a 0, or 99? Since the CHOICE we make will determine which part of the program is called next, we want to be sure that unwanted errors are not caused, so we need to perform a 'range checking' procedure. This is a small routine that checks to see if the number input is within the acceptable range before allowing the program to continue. Here is a sample routine designed to trap an erroneous input.

RANGE CHECKING ROUTINE

```
1 REM ROUTINE
10 LET L = 0
20 FOR L = 1 TO 1
30 INPUT "ENTER 1–9";CHOICE
40 IF CHOICE <1 THEN LET L = 0
50 IF CHOICE >9 THEN LET L = 0
60 NEXT L
70 PRINT "CHOICE WAS ";CHOICE
80 END
```

Many versions of BASIC can make this routine simpler by including a boolean operator in the test like this:

```
10 LET L = 0
20 FOR L = 1 TO 1
30 INPUT "ENTER 1–9";CHOICE
40 IF CHOICE < 1 OR CHOICE > 9 THEN LET L = 0
50 NEXT L
60 PRINT "CHOICE WAS ";CHOICE
70 END
```

These routines also illustrate another point about the INPUT statement. INPUT causes the program to stop and wait for an input from the keyboard. BASIC does not know when the whole number has been entered until the RETURN key has been pressed, so you will also have to remember to press RETURN after entering the number.

A more 'user friendly' approach would be to have the program continue as soon as a valid number had been entered. This is possible using the INKEY$ function. Here, BASIC reads a character from the keyboard whenever INKEY$ is encountered. The program does not stop, however, and will proceed to the next part of the program without pausing. It is usual, therefore, for INKEY$ to be used within loops. The loop to check for a key being pressed can be IF INKEY$ = "" THEN... — in other words, if the key being pressed is 'nothing' (that is, no key is being pressed), go back and check again. A suitable loop for our purposes would be:

```
LET I = 0
FOR I = 1 TO 1
LET A$ = INKEY$
IF A$ = "" THEN LET I = 0
NEXT I
```

The only disadvantage of using INKEY$ is that it returns a character from the keyboard, rather than a numeric. When there is a CASE OF construct, where one out of several choices are made (a multi-conditional branch), it is easier in BASIC to use numbers rather than characters. This is where BASIC's NUM or VAL functions come in. They convert numbers in character strings into 'real' numbers (that is, numeric values, not ASCII codes representing numerals). They are used like this:

```
LET N = VAL(A$) or LET N = NUM(A$)
```

By using the NUM or VAL functions, we can have the program convert inputs, using INKEY$, into numeric variables. This removes the need to use the RETURN key after the number key has been pressed. Out-of-range checking is still advisable, however.

The following program fragment involves two loops, one nested within the other. The inner loop waits for a key to be pressed; the outer loop converts the string to a number and checks that it is within range: