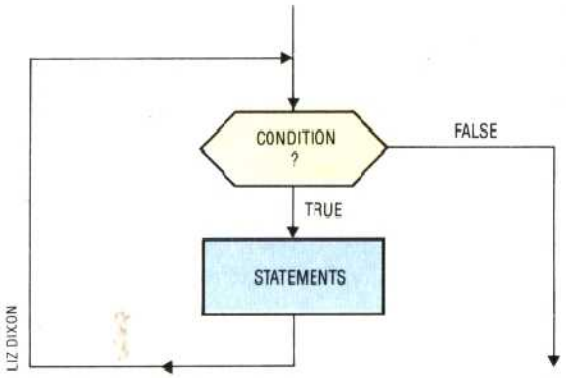
```
250 PRINT "PRESS SPACE-BAR WHEN READY"
260 IF INKEY$ < > "    " THEN GOTO 260
270 GOSUB *START*
```



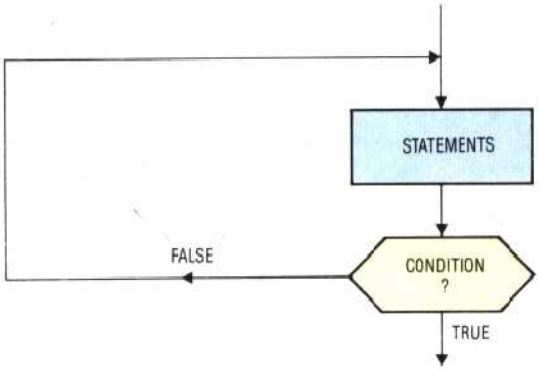
The DO . . . WHILE Control Structure
The loop is repeated as long as the condition is True. The statements may never be executed (if the initial condition is False)

Line 260 says IF INKEY$ is not equal to (< >) a space (" ") THEN go back and check the keyboard again. A slightly more elegant way of writing it would be:

```
250 PRINT "PRESS SPACE-BAR WHEN READY"
260 FOR X = 0 TO 1 STEP 0
270 IF INKEY$ = "    " THEN LET X = 2
280 NEXT X
290 GOSUB *START*
```

In this program fragment the loop (to scan the keyboard) is executed only if the space-bar has not been pressed. If the space-bar has been pressed (i.e. INKEY$ = " ") then the program exits from the FOR . . . NEXT loop to line 290, which is the call to the START subroutine. (NB. We are using 'labels' or names for subroutines. Many versions of BASIC cannot call subroutines by name and you will have to use line numbers instead of labels.)

We haven't encountered STEP before, and this is perhaps an unusual application for it. When using a FOR . . . NEXT loop, STEP allows the 'index' to be incremented in units other than one. FOR I = 1 TO 10 STEP 2 will cause I to have the value 1 on the first pass of the loop, followed by 3,5,7 and 9. The next increment (to 11) will exceed the limit of 10 so the loop will be finished. It is even possible to have the index counting backwards. For I = 10 TO 1 STEP-1 will cause I to count from 10 down to 1. Using STEP 0 is really a clever trick that ensures that the loop will never finish unless X is 'artificially increased' — as in the case of our IF . . . THEN statement.



The REPEAT . . . UNTIL Control Structure
The loop is repeated until the condition becomes True. The statements will always be executed at least once

Another useful control structure, again not directly available in BASIC but easily simulated, is REPEAT . . . UNTIL. Here the condition test comes after the main body of the loop, so the statement or statements in the main body will always be repeated at least once. Look at this 'random number generator':

```
10 PRINT "HIT THE SPACE-BAR"
20 FOR X = 0 TO 1 STEP 0
30 LET R = R + 1
40 IF R > 9 THEN LET R = 1
50 IF INKEY$ = "    " THEN LET X = 2
60 NEXT X
70 PRINT "THE VALUE OF R IS ";R
```

Here, the main body (incrementing the value of R) is always executed at least once since the test to branch out of the loop (IF INKEY$ = " ") does not come until after the increment statement (LET R = R + 1).

Yet another non-essential but useful control structure is that usually called CASE. In BASIC, the CASE structure is implemented using either ON . . . GOTO or ON . . . GOSUB. This is how it works. ON . . . GOTO is a multiple-branching statement that incorporates several IF . . . THEN conditional tests into a single statement. Consider a program fragment that converts the numbers 1 to 7 into the words for the seven days of the week:

```
1050 IF D = 1 THEN GOTO 2020
1060 IF D = 2 THEN GOTO 2040
1070 IF D = 3 THEN GOTO 2060
1080 IF D = 4 THEN GOTO 2080
1090 IF D = 5 THEN GOTO 3000
2000 IF D = 6 THEN GOTO 3020
2010 IF D = 7 THEN GOTO 3040
2020 PRINT "MONDAY"
2030 GOTO*END*
2040 PRINT "TUESDAY"
2050 GOTO*END*
2060 PRINT "WEDNESDAY"
2070 GOTO*END*
2080 PRINT "THURSDAY"
2090 GOTO*END*
3000 PRINT "FRIDAY"
3010 GOTO*END*
3020 PRINT "SATURDAY"
3030 GOTO*END*
3040 PRINT "SUNDAY"
3050 GOTO*END*
```

A more compact way of achieving the same object in BASIC is to use ON . . . GOTO like this:

```
1050 ON D GOTO 2020,2040,2060,2080,
     3000,3020,3040
```

ON . . . GOSUB works the same way, except that the value of the variable determines which subroutine is branched to. Here is a slight modification of the dice program (see page 174) using ON . . . GOSUB to select the appropriate graphics for the dice selected by the RND function:

DECIMAL	BINARY	CHARACTER
32	0 0 1 0 0 0 0 0	=(space)
33	0 0 1 0 0 0 0 1	= !
34	0 0 1 0 0 0 1 0	= "
35	0 0 1 0 0 0 1 1	= #
36	0 0 1 0 0 1 0 0	= $
37	0 0 1 0 0 1 0 1	= %
38	0 0 1 0 0 1 1 0	= &
39	0 0 1 0 0 1 1 1	= '
40	0 0 1 0 1 0 0 0	= (
41	0 0 1 0 1 0 0 1	=)
42	0 0 1 0 1 0 1 0	= *
43	0 0 1 0 1 0 1 1	= +
44	0 0 1 0 1 1 0 0	= ,
45	0 0 1 0 1 1 0 1	= -
46	0 0 1 0 1 1 1 0	= .
47	0 0 1 0 1 1 1 1	= /
48	0 0 1 1 0 0 0 0	= 0
49	0 0 1 1 0 0 0 1	= 1
50	0 0 1 1 0 0 1 0	= 2
51	0 0 1 1 0 0 1 1	= 3
52	0 0 1 1 0 1 0 0	= 4
53	0 0 1 1 0 1 0 1	= 5
54	0 0 1 1 0 1 1 0	= 6
55	0 0 1 1 0 1 1 1	= 7
56	0 0 1 1 1 0 0 0	= 8
57	0 0 1 1 1 0 0 1	= 9
58	0 0 1 1 1 0 1 0	= :
59	0 0 1 1 1 0 1 1	= ;
60	0 0 1 1 1 1 0 0	= <
61	0 0 1 1 1 1 0 1	= =
62	0 0 1 1 1 1 1 0	= >
63	0 0 1 1 1 1 1 1	= ?
64	0 1 0 0 0 0 0 0	= @
65	0 1 0 0 0 0 0 1	= A
66	0 1 0 0 0 0 1 0	= B
67	0 1 0 0 0 0 1 1	= C
68	0 1 0 0 0 1 0 0	= D
69	0 1 0 0 0 1 0 1	= E
70	0 1 0 0 0 1 1 0	= F
71	0 1 0 0 0 1 1 1	= G
72	0 1 0 0 1 0 0 0	= H
73	0 1 0 0 1 0 0 1	= I
74	0 1 0 0 1 0 1 0	= J
75	0 1 0 0 1 0 1 1	= K
76	0 1 0 0 1 1 0 0	= L
77	0 1 0 0 1 1 0 1	= M
78	0 1 0 0 1 1 1 0	= N
79	0 1 0 0 1 1 1 1	= O
80	0 1 0 1 0 0 0 0	= P
81	0 1 0 1 0 0 0 1	= Q
82	0 1 0 1 0 0 1 0	= R
83	0 1 0 1 0 0 1 1	= S
84	0 1 0 1 0 1 0 0	= T
85	0 1 0 1 0 1 0 1	= U
86	0 1 0 1 0 1 1 0	= V
87	0 1 0 1 0 1 1 1	= W
88	0 1 0 1 1 0 0 0	= X
89	0 1 0 1 1 0 0 1	= Y
90	0 1 0 1 1 0 1 0	= Z
91	0 1 0 1 1 0 1 1	= [
92	0 1 0 1 1 1 0 0	= \
93	0 1 0 1 1 1 0 1	=]
94	0 1 0 1 1 1 1 0	= ^
95	0 1 0 1 1 1 1 1	= _
96	0 1 1 0 0 0 0 0	= `
97	0 1 1 0 0 0 0 1	= a
98	0 1 1 0 0 0 1 0	= b
99	0 1 1 0 0 0 1 1	= c
100	0 1 1 0 0 1 0 0	= d
101	0 1 1 0 0 1 0 1	= e
102	0 1 1 0 0 1 1 0	= f
103	0 1 1 0 0 1 1 1	= g
104	0 1 1 0 1 0 0 0	= h
105	0 1 1 0 1 0 0 1	= i
106	0 1 1 0 1 0 1 0	= j
107	0 1 1 0 1 0 1 1	= k
108	0 1 1 0 1 1 0 0	= l
109	0 1 1 0 1 1 0 1	= m
110	0 1 1 0 1 1 1 0	= n
111	0 1 1 0 1 1 1 1	= o
112	0 1 1 1 0 0 0 0	= p
113	0 1 1 1 0 0 0 1	= q
114	0 1 1 1 0 0 1 0	= r
115	0 1 1 1 0 0 1 1	= s
116	0 1 1 1 0 1 0 0	= t
117	0 1 1 1 0 1 0 1	= u
118	0 1 1 1 0 1 1 0	= v
119	0 1 1 1 0 1 1 1	= w
120	0 1 1 1 1 0 0 0	= x
121	0 1 1 1 1 0 0 1	= y
122	0 1 1 1 1 0 1 0	= z
123	0 1 1 1 1 0 1 1	= {
124	0 1 1 1 1 1 0 0	= |
125	0 1 1 1 1 1 0 1	= }
126	0 1 1 1 1 1 1 0	= ~

ASCII
Here is a complete list of the ASCII values between 32 and 126, their binary equivalents, and the characters they represent. The meaning attached to values outside this range varies considerably from machine to machine