Chapter 12: Keyboard Control – Using the Keyboard to Do Things.

This chapter will show you how to make your program respond to the user when a key is pressed (arrows, letters, and special keys) on the keyboard.

Getting the Last Key Press:

The *key* function returns the last raw keyboard code generated by the system when a key was pressed. Certain keys (like control-c and function-1) are captured by the BASIC256 window and will not be returned by key. After the last key press value has been returned the function value will be set to zero (0) until another keyboard key has been pressed.

The key values for printable characters (0-9, symbols, letters) are the same as their upper case Unicode values regardless of the status of the caps-lock or shift keys.

```
# readkey.kbs
print "press a key - Q to quit"
do
    k = key
    if k <> 0 then
        if k >=32 and k <= 127 then
            print chr(k) + "=";
        end if
        print k
    end if
until k = asc("Q")
end
```

Program 73: Read Keyboard

```
press a key - Q to quit
A=65
Z=90
M=77
16777248
&=38
7=55
```

Sample Output 73: Read Keyboard

New Concept

```
key
key()
```

The **key** function returns the value of the last keyboard key the user has pressed. Once the key value is read by the function, it is set to zero to denote that no key has been pressed.

Partial List of Keys

ESC= 16777216		Space= 32			
0=48	1=49	2=50	3=51	4=52	5=53
6=54	7=55	8=56	9=57		
A=65	B=66	C=67	D=68	E=69	F=70
G=71	H=72	I=73	J=74	K=75	L=76
M=77	N=78	O=79	P=80	Q=81	R=82
S=83	T=84	U=85	V=86	W=87	X=88
Y=89	Z=90				
Down Arrow= 16777237			Up Arrow= 16777235		
Right Arrow= 16777236			Left Arrow= 16777234		

See http://qt-project.org/doc/qt-4.8/qt.html#Key-enum for a complete list of key values.

New Concept

Unicode

The Unicode standard was created to assign numeric values to letters or characters for the world's writing systems. There are more than 107,000 different characters defined in the Unicode 5.0 standard.

See: http://www.unicode.org

New Concept

```
asc(expression)
```

The **asc** function returns an integer representing the Unicode value of the first character of the string *expression*.

New Concept

```
chr(expression)
```

The **chr** function returns a string, containing a single character with the Unicode value of the integer *expression*.

Another example of a key press program would be a program to display a letter and to time the user to see how long it took them to press the letter on the keyboard. This program also introduces the **msec** statement that returns the number of milliseconds (1/1000 of a second) that the program has been running.

```
# keymsec.kbs

# get the code for a random character from A-Z
c = asc("A") + int(rand*26)

# display the letter (from the numeric code)
print "press '" + chr(c) + "'"

time = msec                 # get the start time
do                          # wait for the key
     k = key
until k = c
time = msec - time          # calculate how long (in ms)

print "it took you " + (time/1000) + " seconds to find that letter."
```

Program 74: Keyboard Speed Drill

```
press 'C'
it took you 1.833 seconds to find that letter.
```

Sample Output 74: Keyboard Speed Drill

New Concept

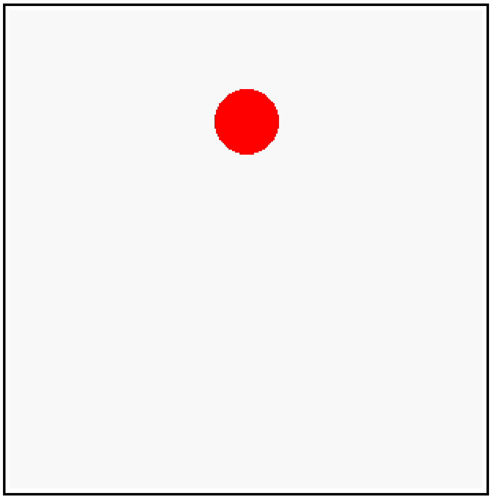
```
msec()
msec
```

The **msec** function returns the length of time that a program has been running in milliseconds (1/1000 of a second).

How about we look at a more complex example? Program 75 Draws a red ball on the screen and the user can move it around using the keyboard.

```
# keymoveball.kbs
# move a ball on the screen with the keyboard

print "use i for up, j for left, k for right, m for down, q to quit"

fastgraphics
clg

# position of the ball
# start in the center of the screen
x = graphwidth /2
y = graphheight / 2
r = 20  # size of the ball (radius)

# draw the ball initially on the screen
call drawball(x, y, r)

# loop and wait for the user to press a key
while true
     k = key
     if k = asc("I") then
          y = y - r
          if y < r then y = graphheight - r
          call drawball(x, y, r)
     end if
     if k = asc("J") then
          x = x - r
          if x < r then x = graphwidth - r
          call drawball(x, y, r)
     end if
     if k = asc("K") then
          x = x + r
          if x > graphwidth - r then x = r
          call drawball(x, y, r)
     end if
     if k = asc("M") then
          y = y + r
          if y > graphheight - r then y = r
          call drawball(x, y, r)
     end if
     if k = asc("Q") then exit while
end while
print "all done."
end

subroutine drawball(ballx, bally, ballr)
     clg white
     color red
     circle ballx, bally, ballr
     color rgb(255,100,100)
     circle ballx+.25*ballr, bally+.25*ballr,
ballr*.50
     color rgb(255,150,150)
     circle ballx+.25*ballr, bally+.25*ballr,
ballr*.30
     color rgb(255,200,200)
     circle ballx+.25*ballr, bally+.25*ballr,
ballr*.10
     refresh
end subroutine
```

Program 75: Move Ball



Sample Output 75: Move Ball

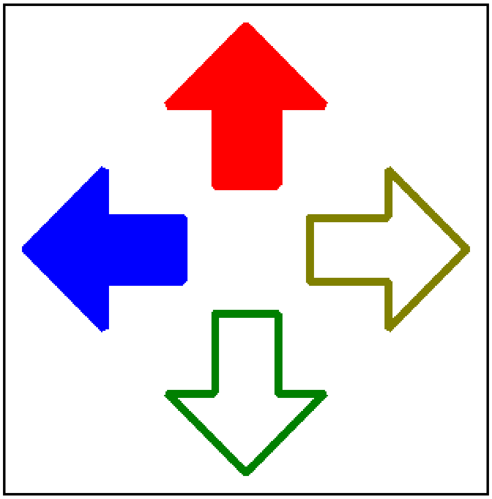
Getting the Currently Pressed Keys

The **key** function in the first half of this chapter returns the last key pressed, even if the user has released the key. We will now see the **keypressed** function that will let us know what keys are being pressed, right now.

```
# keypressarrows.kbs

arrow = { {5, 0}, {10, 5}, {7, 5}, {7, 10}, {3, 10}, {3, 5}, {0, 5}}

ar_down = 16777237
ar_up = 16777235
ar_left = 16777234
ar_right = 16777236
```

```
space = 32

clg white
penwidth 5

print  "press arrow keys on keyboard (even more than one) or space to end"
while not keypressed(space)
     if keypressed(ar_up) then
          color red
     else
          color darkred, white
     endif
     stamp 100,10,10,arrow

     if keypressed(ar_down) then
     color green
     else
          color darkgreen, white
     endif
     stamp 200,290,10,pi,arrow

     if keypressed(ar_left) then
     color blue
     else
          color darkblue, white
     endif
     stamp 10,200,10,1.5*pi,arrow

     if keypressed(ar_right) then
     color yellow
     else
          color darkyellow, white
     endif
     stamp 290,100,10,.5*pi,arrow

end while
```

Program 76: Keys Pressed



Sample Output 76: Keys Pressed

New Concept

`keypressed(key_value)`

The **keypressed** function returns true if the key number is currently being pressed. This statement may be used to see if multiple keys are being pressed at the same time.

See the key function above for a list of common keycodes.

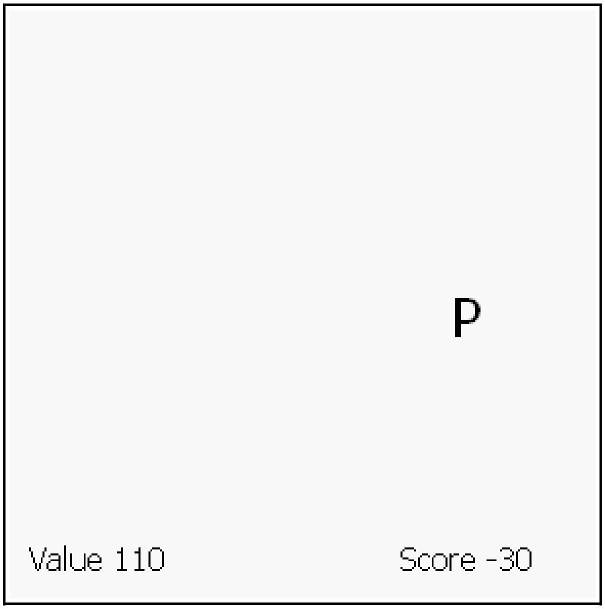
Big Program

The big program this chapter is a game using the keyboard. Random letters are going to fall down the screen and you score points by pressing the key as fast as you can.

```
# fallinglettergame.kbs

speed = .15 # drop speed - lower to make faster
nletters = 10 # letters to play

score = 0
misses = 0
color black

fastgraphics

clg
font "Tahoma", 20, 50
text 20, 80, "Falling Letter Game"
font "Tahoma", 16, 50
text 20, 140, "Press Any Key to Start"
refresh
# clear keyboard and wait for any key to be pressed
k = key
while key = 0
   pause speed
end while

misses = nletters # assume they missed everything
for n = 1 to nletters
   letter = int((rand * 26)) + asc("A")
   x = 10 + rand * 225
   for y = 0 to 250 step 20
```

```
      clg
      # show letter
      font "Tahoma", 20, 50
      text x, y, chr(letter)
      # show score and points
      font "Tahoma", 12, 50
      value = (250 - y)
      text 10, 270, "Value "+ value
      text 200, 270, "Score "+ score
      refresh
      k = key
      if k <> 0 then
         if k = letter then
            score = score + value
            misses-- # didnt miss this one
         else
            score = score - value
         end if
         exit for
      end if
      pause speed
   next y
next n

clg
font "Tahoma", 20, 50
text 20, 40, "Falling Letter Game"
text 20, 80, "Game Over"
text 20, 120, "Score: " + score
text 20, 160, "Misses: " + misses
refresh
end
```

Program 77: Big Program - Falling Letter Game



```
                    P

Value 110        Score -30
```

Sample Output 77: Big Program - Falling Letter Game

Exercises:

Word Search

arrow, asc, capslock, chr, control, key, shift, unicode, keypressed, escape

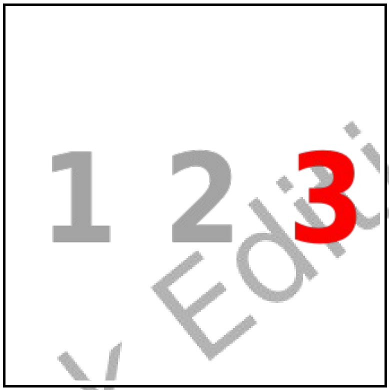
Problems

1. Take Program 74: Keyboard Speed Drill from this chapter and modify it to display ten letters, one at a time, and wait for the user to press that key. Once the user has pressed the correct letters display the total time it took the user.

As an added challenge add logic to count the number of errors and allow a user to retry a letter until they successfully type it.

```
press 'A'
press 'M'
press 'O'
error
press 'U'
press 'X'
press 'V'
press 'K'
press 'C'
press 'Z'
press 'Z'
it took you 15.372 seconds to find
them.
you made 1 errors.
```

2. Create a graphical game like "whack-a-mole" that displays a number on the screen and will wait a random length of time (try 0.5 to 1.5 seconds) for the user to press that number. If they do play a happy sound and display the next, if they miss it or are not fast enough play a sad sound. When they have missed 5 then show them how many they were able to get.



3. Create a piano program using the keys of your keyboard. Wait in a loop so that when the user presses a key the program will play a sound for a short period of time. Assign keys on the keyboard frequencies that correspond to notes on Illustration 10 found on page 52.

4. Use the keypressed function to animate a ball on the screen. You may want to start with Program 75, above.