

Keyboard Input

Introduction

Although most games don't need to accept text from the keyboard, there will be occasions when this would be useful (perhaps entering your name for a high score).

The INPUT statement isn't really up to the job of keyboard input if we want our program to stay in sprite or 3D mode. So, in these next couple of pages we'll have a look at another approach to keyboard input.

The INKEY\$ Statement

The INPUT waits for you to type in as many characters as you want, displaying each on the screen, and stops only once you've pressed Enter.

INKEY\$, on the other hand, is quite a different type of statement. It accepts only one character, the character doesn't get displayed on the screen, and it doesn't even wait for you to hit the key! If you're not pressing a key at the instant the INKEY\$ statement is executed, then you've missed it!

If the INKEY\$ statement executes when no key is being pressed, it returns an empty string; if a key is being pressed, then a string containing that key is returned.

The program in LISTING-Sup001 is a rather optimistic attempt at reading a key and displaying its value.

LISTING-Sup001

Trying to use INKEY\$

```
REM *** Read a key ***
ch$ = INKEY$()

REM *** Display it ***
PRINT ch$

REM *** End program ***
WAIT MOUSE
END
```

Activity Sup004

Try the program given above (*inkey01.dbpro*). If you're really lucky, you might display a character, but you'll have to be fast!

We could give ourselves a better chance if we put the INKEY\$ statement in a loop structure (see LISTING_Sup002).

LISTING-Sup002

Waiting for INKEY\$

```
REM *** Keep trying until a key press is detected ***
REPEAT
  ch$ = INKEY$()
UNTIL ch$ <> ""

REM *** Display the key ***
PRINT ch$

REM *** End program ***
WAIT MOUSE
END
```

Activity Sup005

Try this version of the program (you should have more success this time).

Reading More than One Key

Since we're likely to need to type something a little longer, we need some way of reading more than a single key. Perhaps if we put the code already produced in a loop (see LISTING-Sup003).

LISTING-Sup003

Displaying the Keys Entered

Notice the semicolon at the end of the PRINT statement. This keeps all the output on the same line.

```
DO
  REPEAT
    ch$ = INKEY$()
  UNTIL ch$ <> ""
  PRINT ch$;
LOOP
END
```

Activity Sup006

Try out this latest version.

Any surprises?

The trouble is that the computer can execute the loops many times in a fraction of a second, so the INKEY\$ statement checks the keyboard several times before you can get your finger off the keyboard.

One way round this would be to add in a delay to slow everything down:

```
DO
  REPEAT
    ch$ = INKEY$()
  UNTIL ch$ <> ""
  PRINT ch$;
  WAIT 200
LOOP
```

but this method doesn't work too well since fast typists will be annoyed by the slow speed while slower people are still likely to get repeating keys. A better approach is to wait until no keys are being pressed before starting round the loop again (see LISTING-Sup004).

LISTING-Sup004

Waiting for the Key Press to Stop

```
DO
  REPEAT
    ch$ = INKEY$()
  UNTIL ch$ <> ""
  PRINT ch$;
  REM *** Wait until no keys are pressed ***
  WHILE INKEY$() <> ""
  ENDWHILE
LOOP
END
```

Activity Sup007

Try out this new version.

In fact, this code will be so useful, it's worth creating a function from it which reads a single character from the keyboard:

```
FUNCTION ReadAKey$( )
  REPEAT
    ch$ = INKEY$( )
  UNTIL ch$ <> ""
  WHILE INKEY$( ) <> ""
  ENDWHILE
ENDFUNCTION ch$
```

Notice that the routine doesn't try to display the character read; instead it returns the character. That way anyone using the routine can do whatever they want with the character, rather than be forced to have it appear on the screen.

Activity Sup008

Create the function given above and use it in a program which accepts and displays characters entered, stopping when a question mark is entered.

The logic required is

```
REPEAT
  Read a character
  Display the character
UNTIL the character is a ?
```

Activity Sup009

The trouble with the last program is that the ? itself is displayed on the screen.

Modify your program so that the ? character does not appear.

What happens if you press the Enter key?

Not all keys produce a printable character. For example, in the last Activity we saw that the Enter key appears to have no effect. In fact every key causes some character to be returned. We can prove this by displaying the ASCII code of the character read rather than the character itself. This requires the line:

```
PRINT ASC(ReadAKey$( ));
```

Activity Sup010

Modify your last program to display the ASCII code of each character entered.

What is the ASCII code for the Enter and Backspace characters?

Change the program back to displaying the actual characters entered, rather than their ASCII code.

Modify your program so that the WHILE..ENDWHILE loop terminates when the Enter key is pressed rather than with the question mark.

Creating a Replacement for INPUT

The INPUT statement has four characteristics that we need to include in any routine that reads a string from the keyboard:

- Each character entered also appears on the screen
- Entry is complete when the Enter key is pressed
- The backspace character can be used to delete previously entered characters.
- The complete set of characters entered is returned as a string which can be assigned to a variable.

We'll have to add each of these characteristics to any new routine.

We already know how to stop input when the Enter key is pressed.

Getting each character to appear on the screen may seem to be a done deal too, but if we use our new routine along with sprites or 3D objects, any text we place on the screen will disappear every time the screen is refreshed (see LISTING-Sup005).

LISTING-Sup005

Text Output in Sprite/3D Mode

```
REM *** Use manual screen updating ***
SYNC ON
REM *** Create a 3D object - that way we use a different mode ***
MAKE OBJECT CUBE 1,1
REM *** Update the screen ***
SYNC:SYNC
ch$ = ReadAKey$()
WHILE ASC(ch$) <> 13
  REM *** Display char and update screen ***
  PRINT ch$;
  SYNC
  ch$ = ReadAKey$()
ENDWHILE
END

FUNCTION ReadAKey$()
  REPEAT
    ch$ = INKEY$()
  UNTIL ch$ <> ""
  WHILE INKEY$() <> ""
  ENDWHILE
ENDFUNCTION ch$
```

Activity Sup011

Change your code to match the program in LISTING-Sup005.

Notice that only one character at a time is displayed.

To get round this problem, we can add each character entered to a string and display that string. The updated main section of the program is shown below:

```
SYNC ON
MAKE OBJECT CUBE 1,1
SYNC:SYNC
result$ = ""
ch$ = ReadAKey$()
WHILE ASC(ch$) <> 13
```

```

result$= result$ + ch$
SET CURSOR 100,100
PRINT result$
SYNC
ch$ = ReadAKey$()
ENDWHILE
END

```

Activity Sup012

Modify your program to match the code above.

We're almost there now. All that is left to do is delete a character from the string when the backspace is pressed. The basic logic for this is:

```

IF keypressed is the backspace THEN
    Remove the last character from the result$ string
ELSE
    Add character to result$
ENDIF
Display result$

```

However, we should only delete from the string if it contains at least one character, so, the IF statement's logic needs to be:

```

IF keypressed is the backspace AND result$ contains at least one character THEN
    ...

```

Activity Sup013

Update your program to match the logic above.

HINTS : Use LEN to find out the number of characters in *result\$*.
Use LEFT\$ to delete the last character from *result\$*.

Now we're ready to create a function from our code:

```

FUNCTION ReadAString$(x,y)
    IF x < 0 OR y < 0
        EXITFUNCTION ""
    ENDIF
    result$ = ""
    ch$ = ReadAKey$()
    WHILE ASC(ch$) <> 13
        IF ASC(ch$)= 8 AND LEN(result$)>0
            result$ = LEFT$(result$,LEN(result$)-1)
        ELSE
            result$= result$ + ch$
        ENDIF
        SET CURSOR x,y
        PRINT result$
        SYNC
        ch$ = ReadAKey$()
    ENDWHILE
ENDFUNCTION result$

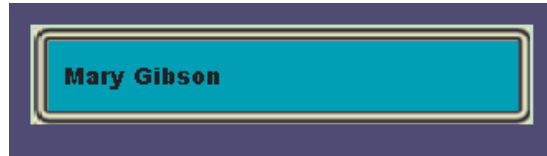
```

Notice that a pre-condition check has been added for the lower values of x and y but none for the upper values. This is because the upper limit depends on the screen resolution, but we could have included some sort of upper limit just to stop ridiculous values being accepted.

By using a box sprite and placing the output at the correct position, we could create the illusion of a text input box as shown in FIG-Sup004.

FIG-Sup004

Entry within an Edit Box



The complete program for this is given in LISTING-Sup006.

LISTING-Sup006

Implementing Keyboard Entry within a Edit Box

We need to tell the program to draw the sprite first otherwise the text ends up under the edit box!

```

REM *** Set up screen ***
SET DISPLAY MODE 1280,1024,32

REM *** Use manual screen updating ***
SYNC ON

REM *** Make sure sprite appears before text ***
DRAW SPRITES FIRST

REM *** Load the edit box sprite ***
LOAD IMAGE "editbox.bmp",1
SPRITE 1,300,300,1

REM *** Update screen ***
SYNC:SYNC

REM *** Change text style and colour ***
SET TEXT FONT "Arial Black"
SET TEXT SIZE 20
INK 0,0

REM *** Read the string ***
name$ = ReadAString$(320,320)

REM *** End program ***
WAIT KEY
END

FUNCTION ReadAString$(x,y)
  IF x < 0 OR y < 0
    EXITFUNCTION ""
  ENDIF
  result$ = ""
  ch$ = ReadAKey$()
  WHILE ASC(ch$) <> 13
    IF ASC(ch$) = 8 AND LEN(result$) > 0
      result$ = LEFT$(result$,LEN(result$)-1)
    ELSE
      result$ = result$ + ch$
    ENDIF
    SET CURSOR x,y
    PRINT result$
    SYNC
    ch$ = ReadAKey$()
  ENDWHILE
ENDFUNCTION result$

FUNCTION ReadAKey$()
  REPEAT
    ch$ = INKEY$()
  UNTIL ch$ <> ""
  WHILE INKEY$() <> ""
  ENDWHILE
ENDFUNCTION ch$

```

Activity Sup014

Modify your program to match the code above. (Make sure you've copied *editbox.bmp* into your folder).

What happens if you enter about 35 characters?

Since it doesn't look too good if we allow the text to extend outside the text box, we could modify the *ReadAString\$()* function to specify a maximum number of characters that can be entered. The new code is highlighted:

```
FUNCTION ReadAString$(max,x,y)
  IF x < 0 OR y < 0 OR max < 1
    EXITFUNCTION
  ENDIF
  result$ = ""
  ch$ = ReadAKey$()
  WHILE ASC(ch$) <> 13
    IF ASC(ch$)= 8 AND LEN(result$)>0
      result$ = LEFT$(result$,LEN(result$)-1)
    ELSE
      IF LEN(result$) < max
        result$= result$ + ch$
      ENDIF
    ENDIF
    SET CURSOR x,y
    PRINT result$
    SYNC
    ch$ = ReadAKey$()
  ENDWHILE
ENDFUNCTION result$
```

The font being used in the example, Arial Black, is a proportional font (that is, some letters are wider than others - w is wider than i), so when you're setting an upper limit for the number of characters, test your code with wide letters - such as W - to check that everything fits within the text box.

Activity Sup015

Modify your program to match the code above and find out the upper limit for *max* when using the sprite image shown.

The only thing missing from the input is a cursor, and although we can't create a flashing one, we could add the illusion of a cursor by displaying the "|" character (ASCII code 124) at the end of the entered string, changing the line

```
PRINT result$
```

to

```
PRINT result$+CHR$(124)
```

and to get the "cursor" to show up at the start we'll also need to add the lines:

```
SET CURSOR x,y
PRINT CHR$(124)
SYNC
```

near the start of the routine.

So, at last, we have a final version of our replacement for the INPUT statement:

```
FUNCTION ReadAString$(max,x,y)
  IF x < 0 OR y < 0 OR max < 1
    EXITFUNCTION ""
  ENDIF
  result$ = ""
  SET CURSOR x,y
  PRINT CHR$(124)
  SYNC
  ch$ = ReadAKey$()
  WHILE ASC(ch$) <> 13
    IF ASC(ch$)= 8 AND LEN(result$)>0
      result$ = LEFT$(result$,LEN(result$)-1)
    ELSE
      result$= result$ + ch$
    ENDIF
    SET CURSOR x,y
    PRINT result$+CHR$(124)
    SYNC
    ch$ = ReadAKey$()
  ENDWHILE
ENDFUNCTION result$
```

Activity Sup016

Change your code to match that given above and test out your program.

Solutions

Activity Sup004

No solution required.

Activity Sup005

No solution required.

Activity Sup006

Even the quickest key press seems to generate about 12 copies of the character!

Activity Sup007

No solution required.

Activity Sup008

```
REPEAT
  ch$ = ReadAKey$()
  PRINT ch$;
UNTIL ch$="?"
WAIT KEY
END

FUNCTION ReadAKey$()
  REPEAT
    ch$ = INKEY$()
    UNTIL ch$ <> ""
    WHILE INKEY$() <> ""
      ENDWHILE
  ENDFUNCTION ch$
```

Activity Sup009

The easiest way to stop the ? appearing on the screen is probably to replace the REPEAT..UNTIL loop in the main section with a WHILE..ENDWHILE structure.

The main section now becomes:

```
ch$ = ReadAKey$()
WHILE ch$ <> "?"
  PRINT ch$;
  ch$ = ReadAKey$()
ENDWHILE
WAIT KEY
END
```

Activity Sup010

To display the ASCII code for each character pressed the main section PRINT statement should be changed as shown below:

```
ch$ = ReadAKey$()
WHILE ch$ <> "?"
  PRINT ASC(ch$), " ";
  ch$ = ReadAKey$()
ENDWHILE
WAIT KEY
END
```

The Enter character has a ASCII code of 13 while the backspace character has the code 8.

To stop when the Enter key is pressed we have to check for the key's ASCII code being equal to 13. The main section now changes to:

```
ch$ = ReadAKey$()
WHILE ASC(ch$) <> 13
  PRINT ch$;
  ch$ = ReadAKey$()
ENDWHILE
WAIT KEY
END
```

Activity Sup011

No solution required.

Activity Sup012

No solution required.

Activity Sup013

The code for the main section is now:

```
SYNC ON
MAKE OBJECT CUBE 1,1
SYNC:SYNC
result$ = ""
ch$ = ReadAKey$()
WHILE ASC(ch$) <> 13
  IF ASC(ch$) = 8 AND LEN(result$) > 0
    result$ = LEFT$(result$,
      LEN(result$)-1)
  ELSE
    result$ = result$ + ch$
  ENDIF
SET CURSOR 100,100
PRINT result$
SYNC
ch$ = ReadAKey$()
ENDWHILE
END
```

Activity Sup014

If you enter too many characters the text overflows the edit box.

Activity Sup015

About 23 is the upper limit before the characters start moving outside the box.

Activity Sup016

No solution required.