INTRODUCTION TO COMPUTER PROGRAMMING

Richard Pierse

Class 5:
Graphics and *Windows* Programming

Windows Programs

Windows programs are very different from the conventional programs we have looked at before now. The most obvious difference is to do with the interface between user and program. *Windows* programs use a ***graphical user interface*** (***GUI***) where, in addition to using the keyboard, the user can click on buttons or select items from lists or from drop-down menus. All *Windows* programs use the same interface, so all will have a familiar feel to the user.

More fundamentally however, *Windows* programs also operate in a completely different way from conventional programs. When the user executes a conventional program, the program starts working immediately. It runs without a break (apart from occasional pauses to request user input), performing its tasks in a set order determined by the program logic. When it has finished its tasks, it terminates automatically.

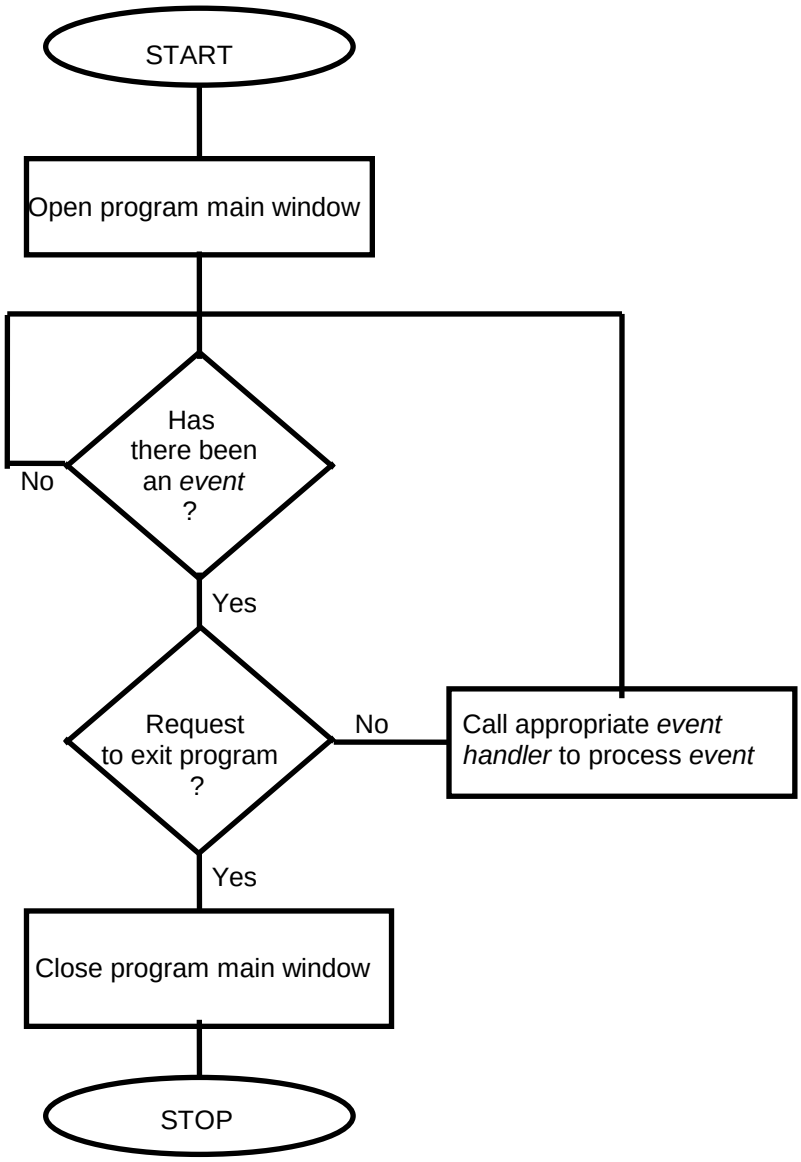
The typical *Windows* program works very differently. When it is first executed, a new window opens on the desktop but then the program stops running and just waits until the user chooses to do something. Tasks are performed at the user's request when a menu item is selected or a button is clicked. It is the user, therefore, rather than the program who decides the order in which tasks are performed and, when each task is completed, the program goes back to waiting again. The program does not terminate automatically but waits until the user requests it to do so.

Windows programs spend most of their time just waiting, doing nothing. While one program is waiting, other *Windows* programs may be executing and the operating system controls this, sharing the CPU between all the programs that are running.

Event-Driven Programming

Windows programs are ***event-driven*** programs. An ***event*** is a user action: a button click, the press of a key on the keyboard or the selection of an item from a drop-down menu or list. The *Windows* program responds to each event by taking an action and the program code that responds to the event is called an ***event handler***, typically, a subroutine or function. There must be an event handler for every possible *Windows* event.

The structure underlying a *Windows* program can be illustrated in the following flow chart. It consists of a loop that runs continuously, checking for events. When an event takes place, it is processed and control returns to the loop. The loop only terminates when the user requests to exit the program, the 'exit program' event.



Opening a Graphics Window

Instead of the standard output window, *Windows* programs use special graphics windows for input and output. Graphics windows have to be opened using an `open` command of the form

```
open "title" for graphics as #hndl
```

where `#hndl` is a handle for the window. Each window must have a unique handle which may be a name or a number. `"title"` is a title to be displayed in the window's title bar.

By default, *Liberty BASIC* always opens a standard output window in addition to any graphics windows that you open. Generally, this is unnecessary since *Windows* programs don't use the standard output window and you can stop the compiler opening one by issuing the command

```
NoMainWin
```

at the beginning of your program.

Controls

Controls are items like buttons, text boxes, list boxes etc. that *Windows* uses for interaction with the user. Controls are actually windows themselves but unlike other windows, their position is anchored to the parent window and they cannot be moved about by the user.

Positioning Controls Within Windows

You can choose to place your windows anywhere on the computer screen. Similarly, when placing controls within windows, they can be positioned anywhere in the main or parent window. Positions are specified in terms of pixels. In standard *VGA* mode there are `640` horizontal pixels and `480` vertical pixels and any position can be specified in terms of two numbers: horizontal (x) and vertical (y) position. For example, position `1,1` is the top left-hand corner of the screen and `640,480` is the bottom right-hand corner. Similarly, a window of width *w* pixels and height *h* pixels with upper left-hand corner positioned at pixel position *x*, *y* can be specified as the quadruple *x, y, w, h*.

Static Text Box Controls

To put text into a graphics window you need to define a special text box. A static text box is a control for text that cannot be changed by the user. A static text box is defined with the command

```
statictext #hndl.ext, ”text”, x,y,w,h
```

- `#hndl.ext` is an identifier for the text box where `#hndl` is the handle of the window in which the text box is to appear
- `”text”` is the text displayed when the window is first opened
- `x,y,w,h` specifies is the position of text box within the window

The text in a static text box can be changed using the command

```
print #hndl.ext, ”replacement string”
```

Command Button Controls

Command buttons are controls that appear in a window and cause some action to be taken when they are clicked. The `button` command declares a button and specifies its name, location and the event handler to be called when the button is clicked.

```
button #hndl.ext, ”text”, [handler], corner, x, y
```

- `#hndl.ext` is an identifier for the button where `#hndl` is the handle of the window in which the button is to appear
- `”text”` is the text to be displayed on the button. It also possible to specify the filename of a bitmap image instead of text
- `[handler]` is the label of the event handler to be called
- `corner` is an identifier specifying the corner to which the button is anchored. It is one of `UL` (upper left), `UR` (upper right), `LL` (lower left) or `LR` (lower right)
- `x, y` is the horizontal and vertical position of the button relative to `corner`.

Note that controls must always be defined before the window in which they are to appear is opened. The correct format is e.g.

```
button #win.but, ”OK”, [ok], UL, 10, 10
open ”myprog” for graphics as #win
```

An Example

The following is an example of a simple *Windows* program, illustrating the use of push buttons and static text. The main window has three buttons labelled “Hello”, “Clear” and “Goodbye”. When the user clicks the “Hello” button, the program executes the event handler labelled `[hi]`, which prints the string “`Hello #n`” to a static text control (where `n` is the number of times the button has been clicked). When the “Clear” button is clicked, the program executes the event handler labelled `[clr]`, which clears the static text control and resets `n`. When the “Goodbye” button is pressed, the program executes the event handler labelled `[bye]` which closes the window and terminates the program.

```
NoMainWin                  ' no standard output window
statictext #win.txt, "", 80,70,150,40
button #win.but1, "Hello", [hi], UL, 25, 25
button #win.but2, "Clear", [clr], UL, 110, 25
button #win.but3, "Goodbye", [bye], UR, 60, 25
open "My program!" for graphics as #win

    n=0
[continue]                 ' start of main program loop
    wait                   ' wait for an event

[hi]                       ' event handler for "Hello" button
    n=n+1
    print #win.txt, "hello #"; n
    goto [continue]        ' return to main loop
[clr]                      ' event handler for "Clear" button
    n=0
    print #win.txt, ""
    goto [continue]        ' return to main loop
[bye]                      ' event handler for "Goodbye" button
    close #win
    end
```

The main loop in this program is the single `wait` command, which causes the computer to pause and wait for an event. When the user clicks on a button, the `button` command causes the appropriate event handler to called (the code labelled `[hi]`, `[clr]` or `[bye]`). After processing the “Hello” or “Clear” buttons, control is returned to the main loop through a `goto` command.

Input Text Boxes

The input text box control allows the user to enter input to the program. The format is

```
textbox #hndl.ext, x,y,w,h
```

The text box can be initialised with default text using the command

```
print #hndl.ext, ”default text”
```

The program reads input from the text box using the command

```
input #hndl.ext, vble
```

where `vble` is the variable to hold the input value, numeric or character as appropriate.

List Boxes

The list box control allows the user to select from a list of items. The format is

```
listbox #hndl.ext, ary$(),[handler],x,y,w,h
```

where `ary$()` is a character array containing the list of items, (which should be initialised before the control is displayed), and `[handler]` is the label of the event handler to call when the user selects an item.

The list box is displayed using the command

```
print #hndl.ext, ”singleclickselect”
```

which specifies that the user can select items by single clicking on the selected item. `”doubleclickselect”` forces double clicking.

The list box event handler can read the selected string using the command

```
input #hndl.ext, item$
```

where `item$` is the character variable to hold the selected item.

Combo Boxes

A combo box control is a combination of a text box and an input box. The user can either type input into a text box window or can select items from a list. The list is generally hidden but can be displayed by clicking on the box's down arrow. The format is

```
combobox #hndl.ext, ary$(),[handler],x,y,w,h
```

The combo box is displayed using the command

```
print #hndl.ext, "selection?"
```

which specifies that the combo box hides the list as soon as an item is selected.

Check Boxes

A check box is a box that can either be selected or deselected by the user. This allows a yes/no or true/false response. The format is

```
checkbox #hndl.ext, "text",[h1],[h2],x,y,w,h
```

where `"text"` is the text associated with the check box, `[h1]` is the event handler to be called when the user checks the box and `[h2]` is the event handler to be called when the user unchecks the box.

Radio Buttons

Radio buttons (or option buttons) are similar to check boxes except that they group choices that are mutually exclusive. Selecting one button from the group automatically deselects all the others (as with buttons on old car radios). All radio buttons in a window form a single group so that one and only one can be selected. The format is the same as for check boxes:

```
radiobutton #hnd.ext, "txt",[h1],[h2],x,y,w,h
```

Liberty BASIC Graphic Commands

Liberty BASIC has several of its own graphics commands that work within graphical windows. These are specific to *Liberty BASIC* and won't work in other versions of *BASIC*. However, they are very easy to use. Here is a list of the main commands. Have fun!

```
print #hndl, "fill green"  'fills window with colour green
print #hndl, "up"  'lift pen from the page - turns off drawing
print #hndl, "down"  'put pen on the page - turns on drawing
print #hndl, "color red"  'set pen colour to red
print #hndl, "goto 20, 40"  'draw line to 20,40
print #hndl, "line 2,6,7,9"  'draw line from 2,6 to 7,9
print #hndl, "box 100, 100"  'draw box  size 100x100
print #hndl, "backcolor blue"  'set background colour
print #hndl, "boxfilled 5, 9"  'draw box in backcolor
print #hndl, "circle 100"  'draw circle radius 100
print #hndl, "circlefilled 100"  'draw filled circle
print #hndl, "ellipse 10,20"  'draw ellipse 10x12
print #hndl, "ellipsefilled 10, 20"  'draw filled ellipse
loadbmp "pic", "c:\logo.bmp"  'load bitmap file
print #hndl, "drawbmp pic 8,10"  'draw bitmap at  8,10
```

To see some of these commands in action, have a look at some of the example files in the *Liberty BASIC* directory such as: `buttons1.bas`, `boxes.bas`, `mandala.bas`, `graphics.bas`, `spaceshp.bas`.

A Graphics Example

The following program modifies the earlier simple *Windows* program to draw boxes and circles of increasing size making use of some of the simple graphics commands listed above.

```
NoMainWin
button #win.but1, ”Square”, [square], UL, 25, 25
button #win.but4, ”Circle”, [circle], UL, 100, 25
button #win.but2, ”Clear”, [clr], UL, 175, 25
button #win.but3, ”Bye”, [bye], UL, 250, 25
open ”My program!” for graphics as #win
n1=0
n2=0
[continue] ' start of main program loop
   wait    ' wait for an event

[square]       ' event handler for “Square” button
   print #win, ”goto ”; 70+n1; ” ”; 70+n1
   print #win, ”down”
   print #win, ”backcolor red”
   print #win, ”boxfilled ”; 100+n1; ”, ”; 100+n1
   print #win, ”up”
   n1=n1+5
   goto [continue]  ' return to main loop

[circle]       ' event handler for “Circle” button
   print #win, ”goto ”; 100+n2; ” ”; 100+n2
   print #win, ”down”
   print #win, ”backcolor green”
   print #win, ”circlefilled ”; 40+n2
   print #win, ”up”
   n2=n2+5
   goto [continue]  ' return to main loop

[clr]      ' event handler for “Clear” button
   print #win, ”cls”
   n1=0
   n2=0
   goto [continue]  ' return to main loop

[bye]
   close #win
   end
```