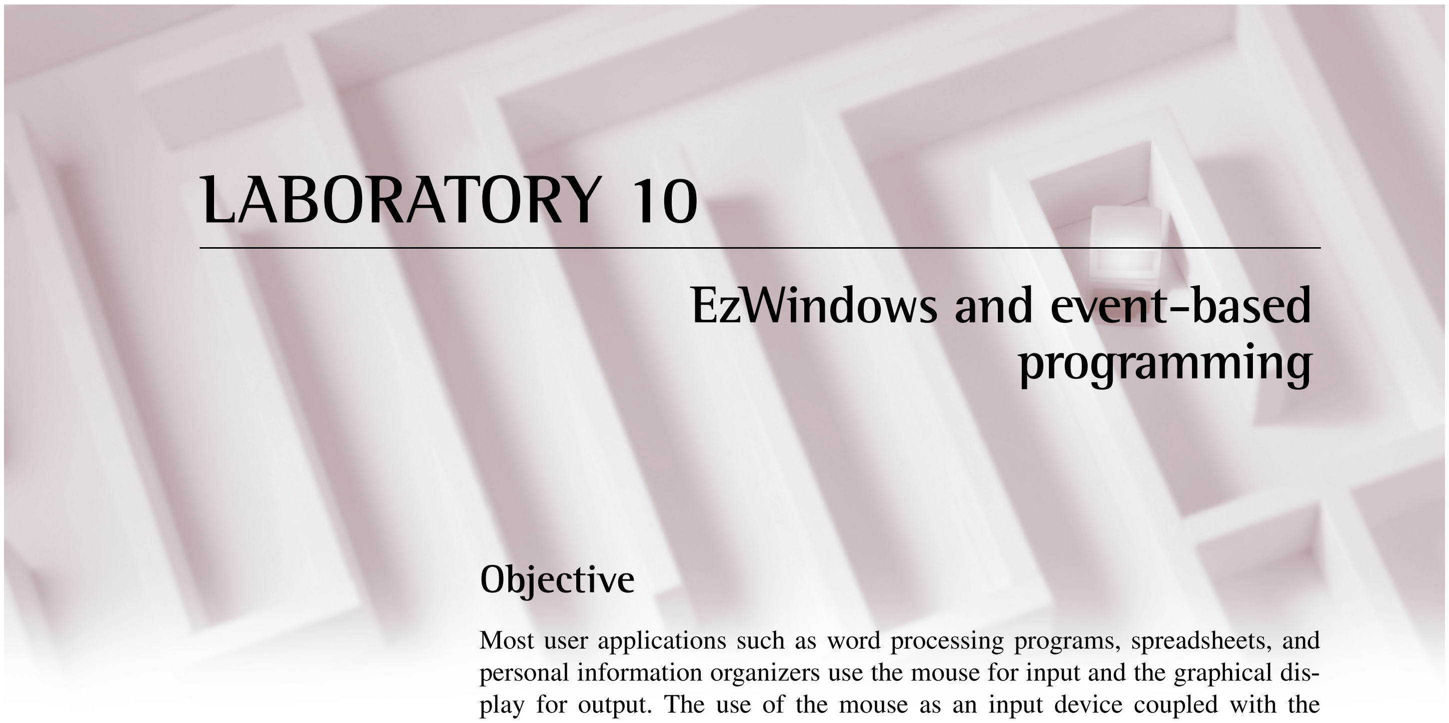


LABORATORY 10

EzWindows and event-based programming

Objective

Most user applications such as word processing programs, spreadsheets, and personal information organizers use the mouse for input and the graphical display for output. The use of the mouse as an input device coupled with the ability to display graphical images has reduced the complexity of using these powerful programs. The objective of this laboratory is to become familiar with EzWindows, a simple interface for using the mouse and displaying graphical objects.

Key Concepts

- Event-based programming
- Callbacks
- Mouse events
- Timer events
- Bitmaps
- Coordinate system

10.1 GETTING STARTED

- Using the procedures in the introductory laboratory handout, create the working directory `\cpplab` on the appropriate disk drive and obtain a copy of self-extracting archive `lab10.exe`. The copy should be placed in the `cpplab` directory. Execute the copy to extract the files necessary for this laboratory.
- Many of the activities that are performed in the laboratory can be done in groups but you should work the exercises yourself.

10.2 EZWINDOWS COORDINATE SYSTEM

Before exploring the EzWindows API further, we need to revisit the coordinate system for positioning objects. EzWindows uses the metric system for specifying both the position of an object and its size. For example, the EzWindows declaration

```
SimpleWindow TestWindow("Sample Window",
 10.0, 5.0, Position(4.0, 4.0));
```

creates a window labeled “Sample Window” that is 4 centimeters from the left edge of the screen and 4 centimeters from the top edge of the screen. The window is 10 centimeters in length and 5 centimeters in height. The prototype of this constructor for the `SimpleWindow` class is

```
SimpleWindow(string WindowTitle = "Untitled",
 float Width = 8.0, float Height = 8.0,
 const Position &WindowPosition
 = Position(0.0, 0.0));
```

Notice that the object specifying the position of the window relative to the upper-left corner of the screen is an instance of the class `Position`. In earlier laboratories, the position of a window object was specified by separate x- and y-coordinates. We did not use the `Position` class in order to keep the number of classes you were dealing with to a minimum. Now that you have a better understanding of objects and object-oriented programming, it makes sense to use the most appropriate representation. It is convenient and natural to encapsulate a window location as a single entity. Consequently, EzWindows contains a class `Position` that holds the logical window coordinates of a window object. The class definition of `Position` is given below.

```
class Position {
    public:
        Position(float x = 0.0, float y = 0.0);
        int GetXDistance() const;
        int GetYDistance() const;
        void SetXDistance(float x);
        void SetYDistance(float y);
    private:
        float XDistance;
        float YDistance;
};
```

Henceforth, we will use a `Position` object to specify the position of a `SimpleWindow` object on the screen. Similarly, we will use a `Position` object to specify the position of EzWindow objects within a `SimpleWindow` object. The methods previously used still work, but they do not encapsulate the notion of a position as well.

10.3 BITMAPS

Most window systems have facilities for displaying images. Bitmap files are recognizable by their `bmp` file extension. For example, Microsoft provides the bitmap `brick.bmp` in its Windows operating system for tiling the background. EzWindows provides a class for displaying and manipulating graphical images in bitmap format.

The following is a partial list of EzWindows `BitMap` class public member functions:

`BitMap::BitMap(SimpleWindow &w)`
: Creates a `BitMap` object with `BitMapStatus NoBitMap`. The object is associated with window `w`.

`BitMap::BitMap()`
: Creates a `BitMap` object with `BitMapStatus NoBitMap`. The object is not associated with any window.

`BitMapStatus BitMap::Load(const **string** &Filename)`
: The file whose name is pointed to by string `Filename` is used to set the bitmap. If the file contains a valid bitmap, the status of the object is set to `BitMapOkay`; otherwise, the status of the object is set to `NoBitMap`.

`void` `BitMap::SetWindow(SimpleWindow &w)`
: Associates the bitmap with window `w`. The `BitMapStatus` of the bitmap is set to `NoBitMap`.

`bool` `BitMap::Draw()`
: Attempts to display the bitmap object to the associated window. The `BitMapStatus` of the object must be `BitMapOkay` for the display to be successful. If the bitmap is displayed, the function returns **`true`**; otherwise, the function returns **`false`**.

`bool` `BitMap::Erase()`
: Overwrites the bitmap on the display by drawing a white rectangle of the same size. If the bitmap is successfully erased, the function returns **`true`**; otherwise, the function returns **`false`**.

`bool` `BitMap::IsInside(`**`const`**` Position &p) `**`const`**
: Returns **`true`** if position `p` lies within the bitmap; otherwise, the function returns **`false`**.

`BitMapStatus BitMap::GetStatus() `**`const`**
: Returns the current `BitMapStatus` value associated with the object.

`float` `BitMap::GetXPosition() `**`const`**
: Returns the distance from the center of the bitmap to the left edge of the associated window. The distance is in centimeters.

`float` `BitMap::GetYPosition() `**`const`**
: Returns the distance from the center of the bitmap to the top edge of the associated window. The distance is in centimeters.

`float` `BitMap::GetWidth() `**`const`**
: Returns the width of the bitmap in centimeters.

float BitMap::GetHeight() **const**
    Returns the height of the bitmap in centimeters.

void BitMap::GetSize(**float &Width, float &Height**) const
    Returns the width and height of the bitmap in centimeters.

void BitMap::SetPosition(**const** Position &p)
    Sets the position of the bitmap to p.

Position BitMap::GetPosition() **const**
    Returns the position of the center of the bitmap.

Important! The bitmap class is not positioned by specifying its center. Rather, a bitmap is positioned by specifying the coordinate of the upper-left corner. See Figure 10.1.

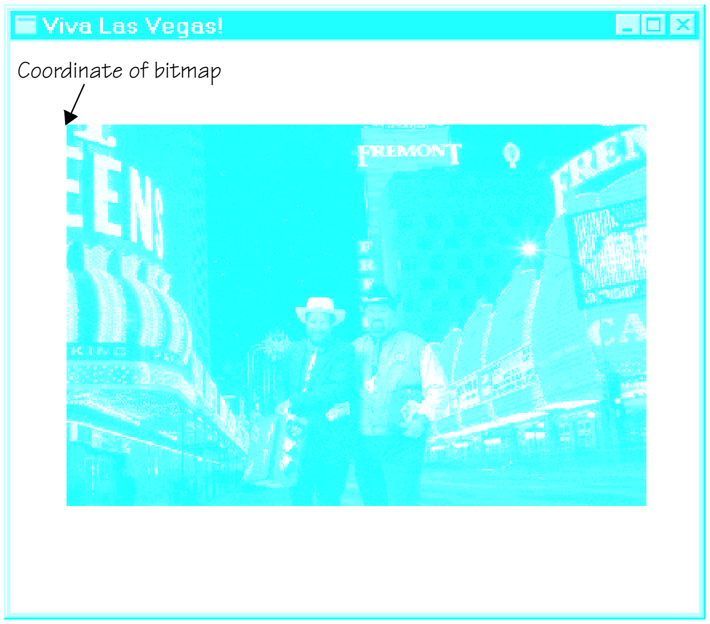
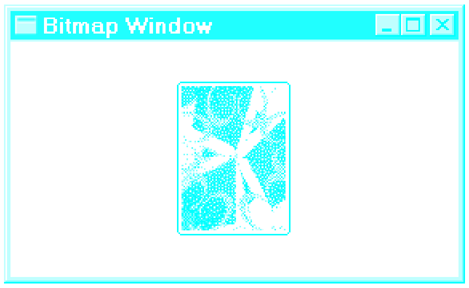


Figure 10.1 *Specifying the position of a bitmap.*

Let's examine a program that creates a window and displays a bitmap.

- Open the project `bitwin.dsw`.

The program creates a window that looks like the following:



- In the project window, double-click on the `bitwin.cpp` file.
- Examine the code. Try to answer the following questions. Experiment with the code to answer any questions that seem difficult to answer.
  - Why does the `BitMap` constructor take a `SimpleWindow` as a parameter?
  - Why call the `assert()` function to check to make sure that the `BitMap` is okay?
  - What happens if the position of the `BitMap` is not set before it is drawn?
  - Is it possible to remove the image from the window without closing the window? Why might that be useful?

✎ ▪ Discuss your conclusions with a laboratory instructor. ✔

- To solidify the previous concepts, modify the program in the following manner:
  - Change the program to load the bitmap `hazard.bmp`.
  - Change the program so that the image is right justified in the window (i.e., the right edge of the image is against the right edge of the window).
  - Change the program to reposition the image after it has been drawn. Reposition the image so that it is left justified in the window. What happens if the original image isn't erased before setting a new position and redrawing?

✎ ▪ Demonstrate your modified program to a laboratory instructor. ✔

- Close the `bitwin.dsw` project.

10.4 MOUSE EVENTS

One of the most powerful tools used to manipulate applications in windows is the mouse. The mouse provides a mode of interaction that is easy to use, is easy to understand, and frees the user from having to remember obscure command names. EzWindows provides a simple facility for using the mouse. Instead of declaring a mouse object, EzWindows allows the user to specify a function to call when a mouse click occurs inside an EzWindow. The `SimpleWindow` declaration of the member function for registering a callback for a mouse event is

```
void SetMouseClickCallback(MouseCallback f);
```

where `MouseCallback` is the `typedef`

```
typedef int (*MouseCallback)(const Position &);
```

When the mouse is clicked, the callback function is called with the coordinate of the mouse pointer as a parameter. The callback function can use the coordinate to determine if the mouse is pointing at some object on the screen.

Let's examine a program that demonstrates the use of mouse callbacks.

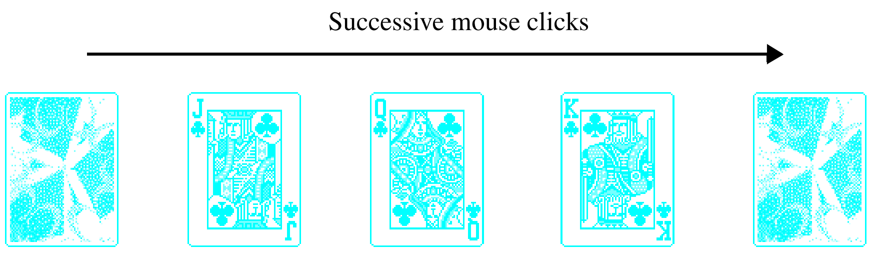
- Open the project `mouse.dsw`.
- In the project window, double-click on the `mouse.cpp` file.
- Examine the code. Try to answer the following questions:
  - — One of three cards is displayed when the mouse is clicked on the back of the card. The three possibilities are the jack, queen, and king. Which face card will appear when you click the mouse on the back of the card?
  - — How does the `BitMapWindow` know which function to call when a mouse event occurs?
  - — Run the program. Click on the card and see what happens. Exit the program. Run the program again. Click in the window, but not on the

card. What happens? Figure out why. If you need help, ask your laboratory instructor.

- Modify the program so that on successive mouse events, a different card appears. The actions performed with each mouse event should proceed as follows. Start with the card back showing. Then

  1. The jack appears.
  2. The queen appears.
  3. The king appears.
  4. The card back appears.
  5. Repeat steps 1 through 4.

  The following diagram illustrates the sequence in which the cards should appear.

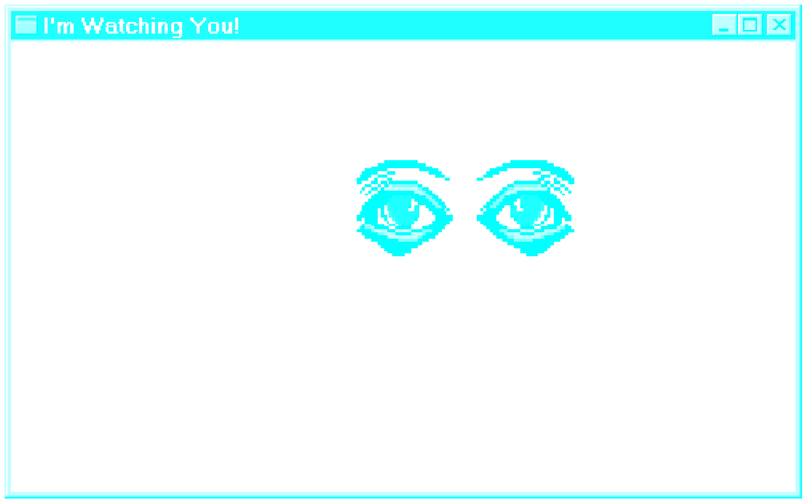


- Hint: You do not need to write additional functions to solve this problem. You can insert `SetMouseClickCallback` calls in existing functions to change the function that will next get called when the mouse is clicked. Think about using this approach to control which card to display next.
- Demonstrate the modifications to a laboratory instructor. ✔
- Close the `Mouse.dsw` project.

10.5 TIMER EVENTS

EzWindows provides a facility for performing actions at a predetermined time or interval. By setting the amount of time that must elapse before a specified action occurs and specifying its callback function, a timer event is setup. A timer event is a dispatch message from the window that invokes a callback function when a specified amount of time has elapsed. Timer events are useful for features such as animation and game timers.

Let's examine a program that has timer events. The following program displays eyes in random locations within the window. When a timer event occurs, the picture is erased, a new location is chosen, and the picture is redrawn.



- Open the project `tictoc.dsw`.
- Run the program. Get a feel for what the program does.
- Terminate the program and return to the IDE.
- In the project window, double-click on the `timer.cpp` file.
- Examine the code. Try to answer the following questions:
  - — Where is the timer callback set?
  - — How much time elapses between each timer event?
  - — Why is member function `StopTimer()` called in the `ApiEnd()` function? What happens if member function `StopTimer()` isn't called?

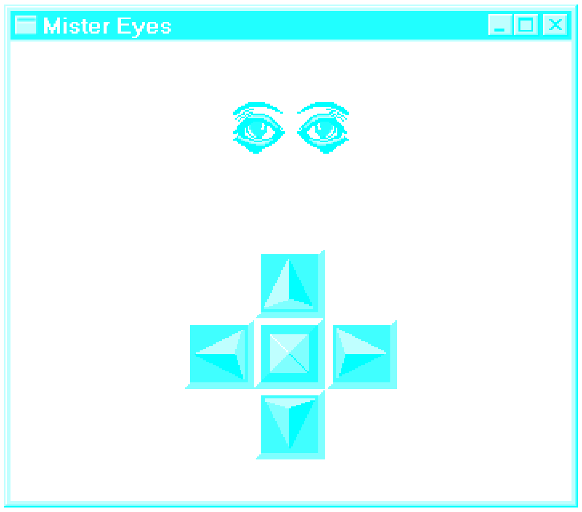
— What units of time specify the elapsed time between timer events?

- Experiment with the code to discover the answers to the above questions.
- ✎ Discuss your answers with the laboratory instructor. ✔
- Make the following modifications to the program:
  - — Change the amount of time elapsed between timer events to five seconds.
  - — Change the amount of time elapsed between timer events so that the picture of the eyes is difficult to follow.
- ✎ Demonstrate your modifications above to a laboratory instructor. ✔
- Close the `tictoc.dsw` project.

10.6 BRINGING IT ALL TOGETHER

Now let's do an exercise that uses all the skills you used in the previous exercises.

The following program displays an EzWindow with a pair of eyes and a set of directional buttons. When a button is pressed, the eyes look in the direction indicated by the picture on the button.



- Open the project `eyes.dsw`.
- Run the program. When you are comfortable with what it does, terminate the program and return to the IDE.

- In the project window, double-click on the `eyes.cpp` file.
- Examine the code. Try to answer the following questions:
  - — The buttons are displayed via a set of `BitMap` objects. In the `ApiMain()` function, why is the `SetWindow()` member function called for each `BitMap`?
  - — In which function is the mouse callback setup?
  - — What function draws the initial picture of the eyes?
  - — What function draws the buttons?
- ✎ Discuss your answers to the preceding questions with your laboratory instructor. ✔
- Complete the implementation of the program so that when a button is pressed the eyes only temporarily look in the designated direction.
- Write the code to complete function `LookStraight()`. It should change the direction in which the eyes are looking to straight ahead.
- Modify function `PressButton()` so that function `LookStraight()` gets called after `LookTime` seconds have elapsed.
- ✎ Demonstrate your modified program to a laboratory instructor. ✔
- Close the `eyes.dsw` project.

10.7 FINISHING UP

- Copy any files you wish to keep to your own drive.
- Delete the directory `\cpplab`.
- Hand in your check-off sheet.