the menu bar shown in this example. What makes this concept so powerful is the way in which Desktop (the operating system) manages the objects created by an application program. Once the code above is executed, all re-displaying and housekeeping is done automatically by Desktop and its companion Finder.

How does the running program link up with the pull-down menus? This is where the combination of the Finder and the resource file come in again. I illustrate this with another piece of abbreviated code taken from the cs561 example and shown in Figure 7.

This fragment of code shows how the pull-down menu items are connected to a routine for carrying out the action indicated by the menu bar. The user points to one of the menu titles in the menu bar and presses the mouse button. This returns a value for theMenu. The user then drags the mouse to select an item; when the button is released, the value of theItem is returned. The code in Figure 7 shows how these values are used to select one of many possible actions via a nested pair of case statements. An abbreviated piece of code for obtaining these events is shown in Figure 8.

```
case theMenu of
    Do_1tMenu: case theItem of
        1: { code for "Compile" goes here };
        2: { code for "Link" goes here };
        3: { code for "Run" goes here };
    end; {DO IT}
end; {Select a Menu}

Design_1tMenu: case theItem of
    1: { code for "Outline" };
    2: { code for "Edit" };
    end; {Design It}
end;

highlightMenu(0);

Figure 7. Program fragment to link up running program with the pull-down menus.
```

```
repeat
    System Task;
    temp := GetNextEvent( An Event, My Event );
    case MyEvent.What of
        MouseDown: case FindWindow( MyEvent.Where,Which);
            InMenuBar: DoCommand;
            InSysWindow: SystemClick;
            end; {MouseDown}
        OtherEvents: {other events handled here}
    end; {MyEvent.What}
until DoneFlag;

Figure 8. Program fragment to obtain the events in Figure 7.
```

Comments. The ideas and techniques incorporated into the Macintosh software are admirable, but I am struck by the boldness of Apple Computer. Programming a Macintosh is not easy for conventional programmers, and this has contributed to the slow appearance of Macintosh software. Does Apple expect everyone to change? Do they think that a technical achievement like the Macintosh is inducement enough to persuade programmers to wade through 800 pages of Inside Macintosh merely to discover the merits of object-oriented programming and Desktop?

The Macintosh software is unconventional, non-standard, and probably horribly non-portable. The dialect of Pascal used is related to UCSD Pascal, but there is little hope of separating it from the Macintosh ROM routines. Yet, the software is nearly a work of art: all objects are organized in a logical and concise manner. The notion of a resource file is powerful and will most likely be emulated by programmers everywhere. Although we did not show it here, Desktop and Finder are designed to accommodate changes in memory size, screen resolution and tint, I/O devices, etc. I suspect the system code will prove to be very easy to maintain and enhance throughout its lifecycle.