A Third-Semester GUI Programming Course

W. Douglas Maurer
Department of Electrical Engineering and Computer Science
The George Washington University
Washington, DC 20052

We describe a new curriculum for the first two years of an undergraduate program in which the students learn full-fledged graphical-user-interface (GUI) programming in the third semester. By full-fledged we mean that the students are expected to use object-oriented techniques in C++ or Java and construct data structures that are sufficient to support a simple word processing, spreadsheet, or data base program written for a GUI. Furthermore this curriculum does not require any computer experience on the part of first semester students. The usual CS 1 and CS 2 are presented in the first and second semesters, except that they are done in C++, and an unusual amount of effort has been expended (as described in another recent paper) to introduce almost all of C++, and almost all of data structures, in one course in such a way that the two subjects support and complement each other. In addition, there is in the first semester, for those who need it, an introductory course in the use of a GUI, including word processing, spreadsheet, data base, and paint programs.

We then proceed to the GUI programming course, which is based around a semester-long project, involving as many GUI features as we can introduce in fourteen weeks. This includes one week for the midterm and one week for the final examination, leaving twelve instructional weeks, each of which involves two instructional units, as outlined below (Units 1 through 24). The students have a choice of developing a simple word-processing system, a simple spread sheet system, or a simple data base system. The instructional units are as follows:

Unit 1. Choose an internal format for your data. Note that the internal format is not what appears on the screen; that is the window format (discussed later).

For a word-processing program, the internal format represents the document to be processed. There is the text, and there is also the formatting (font, size, and style). Since formatting typically applies to long stretches of text, the student has to devise a coding method for these. Special characters, such as optional hyphens and paragraph marks, require codes. It will have to be decided whether the text is internally organized by lines, by paragraphs, or by an entire document.

For a spreadsheet program, the internal format represents the data currently in the spreadsheet. Since spreadsheets are typically organized as sparse matrices (that is, most of the cells are empty), the student will have to devise a sparse matrix representation. Each non-empty cell will have to be marked as a formula cell or a data cell. Data cells can contain strings, which are easy to represent. They can also contain integers or real numbers, and it will have to be decided whether to represent these in binary or in character code format.

For a data base program, the internal format represents the records in the data base. There will be a group of these records which may be organized in various ways (as a linked list, a doubly linked list, a hash table, etc.). Each record in turn is organized into fields, but these have formats which may be changed by the user at run time, unlike those of an ordinary record. A universal field representation will have to be chosen.

Multiple open files in any of these programs (more than one word processing document, spread sheet, or data base open at one time) is actually quite simple; there is a short linked list of all open files with pointers to internal descriptions for each.

Unit 2. Choose an external format for your data (that is, when it is on disk). Note that the disk data format is not necessarily the same as the internal data format; in particular, if there are pointers, you don’t want to write out the actual pointers on disk.

For a word-processing program, the student will have to decide whether to save space on disk by keeping the data in compressed form. A typical compressed form involves four-bit codes for each of the 15 most common characters (space, comma, period, and lower case *e*, *t*, *a*, *o*, *i*, *n*, *s*, *h*, *r*, *d*, *l*, and *u*) and twelve-bit codes for the others (the four-bit code 0 followed by the eight bits of the normal ASCII code).

For a spreadsheet program, the non-null cells are kept as records in some order (actually, the order shouldn’t matter). Again, each cell is specified as either a formula cell or a data cell. Since these cells have different sizes on disk, an indication, at the start of each cell, giving its total size should be included in the external format, although this will not be necessary internally. There will need to be either an end-of-file flag or a special count variable indicating the total number of non-null cells in the file.

For a data base program, there will be an external description of the fields in a data base, followed by records containing the data. If these records contain integers or floating point numbers in internal format, care must be taken that machine alignment rules are followed when intermixing these with character strings.

When there are multiple open files, external formats, of course, apply to only one of these at a time.

Unit 3. Write an output function that writes your data, in your internal format, onto a file, in the given external format. Assume, for the moment, that the name of the file is given as a parameter.

For a word-processing program, this may involve conversion of the file to the given compressed format. It should not be difficult to check whether a character is one of the special 15 characters, and to write out a four-bit code if it is. Otherwise, a four-bit code and an eight-bit code are written out. A buffering routine for four-bit data is easy to write once the principles of it are established.

For a spreadsheet program, it may be necessary to count the non-empty cells first, if such a count is being kept on disk. Each cell is kept in any one of a number of formats, and an indication must be stored on disk, for each cell, as to what format it is in.

For a data base program, the internal format of a record may well involve pointers to its various fields. We must remember not to write out the pointers, but only the fields.

In any of these cases, if the internal format is in error, produce a system-level error message and quit.

Unit 4. Write an input function that reads your data from a file with a given name, in the given external format, and puts it in the machine in your internal format. Again assume that the name of the file is given as a parameter. Test the input function by reading one file, writing another one, and then checking, within another program, whether the two files are the same.

For a word-processing program, it may be necessary, if the file is not organized by lines, to organize it in this way as it is read in. If this is done, then formatting will probably need to be determined and kept separately for each line. The length of a line, of course, depends on the formatting information. For a spreadsheet program, each cell is read in, its format determined, and its data inserted into dynamic storage. For a data base program, the pointers that were taken out of records when we wrote them to disk have now got to be put back in.

As for input, produce a system-level error message and quit if the input is in the wrong format.

Unit 5. Here we handle errors in a more realistic manner than in the previous two units.

Learn how to use the resource editor (sometimes called a form designer) for dialogs. This allows a user to design a dialog box, using graphical tools like those in a paint program. When the dialog box is designed, it is given a name of some kind, by which it may be referenced later.

Set up an alert meaning that the given file is in the wrong format for reading. Learn how to call the alert and use it in your program. Wait for the user to click OK and then terminate.

Unit 6. Learn how to use the graphical function package that comes with your system. Set up another dialog, which is an opening screen for your application, and using as many of the graphical functions as you can. It should have a Continue button. Call this dialog at the start of your program; wait for the user to click Continue, and then continue with your program. The design of this should indicate what kind of an application this is.

Unit 7. Up to now we have assumed, when reading input and writing output, that the file name is given as a parameter. This is not realistic, and in this unit and the next one we learn how it is really done.

Learn how to use the standard open-file dialog. Use this dialog just before you read your file. The dialog will return the name of the file to be opened; use this as a parameter to your file-open routine.

Unit 8. Learn how to use the standard save-file dialog. Use this dialog just before you write your file. The dialog will return the name of the file to be saved; use this as a parameter to your file-save routine. Check whether the standard save-file dialog asks the usual question about replacing an existing file with the same name.

Standard open-file and save-file dialogs are available with every GUI system, but non-standard ones can often be purchased or are available as freeware or shareware. This may be a good time to compare and contrast the features of various such dialogs.

Unit 9. Learn how to set up a window for your file. At this point in the project, the data should be short enough that it will fit entirely on the screen. After you have read the data in, open a window and then draw your data on the window. Make this data drawing into a function which can be called when you want to re-draw your data after making changes to it. For now, however, just put the window up on the screen before saving the file.

For a word-processing program, the window will have certain basic characteristics, such as the margins on each side, single or double spacing, and whether the data is left justified, fully justified, centered, or whatever. Once these are known, the document to be processed can be drawn on the window. Produce another alert box, for the moment, to warn the user if the document is too large for the window.

For a spreadsheet program, we will assume for the moment that there are only a few columns and a few rows. Again the window will have certain basic characteristics, such as a width for each column. For a formula cell, you will have to display the results of the formula. This in general is a large computation, but for the moment we can assume only one operator, namely +, so as to make it easier.

For a data base program, display just one record, having Next and Previous buttons in it. Clicking on one of these buttons will cause the next record, or the previous record, to appear on the screen. Each record will have to be drawn by a general function which scans the format of the record and draws it accordingly.

Unit 10. Before we set up a menu, we will set up a dialog box with several buttons, which does the same thing as a menu. These might be: Open a file; Save the file; Quit; etc. Put this up at the beginning of your program and keep looping back to it.

At this point GUI systems differ as to how they detect mouse clicks on a window. Some GUI systems will just give you back the X and Y coordinates of a mouse click on a window, and then you have to decide what the user is clicking on. Others will let you define objects in the window, and associate actions with clicking on these objects.

Unit 11. Write a function to display in your window only part of your data, as given by parameters. Then implement a resizing box in your window, which resizes both horizontally and vertically, as usual. Also implement dragging the window to various places on the screen.

For a word-processing program, note that resizing the window does not mean changing the margins. The margins remain the same, but only part of each line is displayed. For a spreadsheet program, resizing the window may mean that more or fewer columns are displayed on the screen, and also that the last column may be only partially displayed. For a data base program, resizing the window may mean that the single record on the screen is only partially displayed.

Unit 12. Put scroll bars in your window and implement those.

For a word-processing program, the display function must now be extended (if it has not been already) to display only certain lines of the text. Which lines will be displayed depends on the position of the elevator in the window. There are two possible conventions for what happens when the elevator is at the bottom of the window — either the last few lines are displayed, or only the last line, at the top of the window, followed by blank space. Horizontal scrolling is optional here, for the moment.

For a spreadsheet program, both horizontal and vertical scroll bars are needed, and the sparse matrix should be extended potentially indefinitely in both directions.

For a data base program, we now change the display routine so that, instead of displaying only one record at a time, it might display more than one. All the records in a file are treated as if they followed one another vertically, like text in a word processing file. Vertical scrolling is then handled much as in a word processing program; horizontal scrolling is again optional.

Unit 13. Extend the dialog box of unit 10 above to allow the opening of several files at once. Now draw several windows at the same time on the screen. This is done by drawing the one at the back first, then the others in order, with the one at the front being drawn last.

Now implement the convention that, if you click on an inactive window, it becomes active. This means that it comes to the front; all the windows from the former position of this one up to the front have to be redrawn. Add to your dialog box a Close button, to close the active window. In this case, again all the other windows must be redrawn in order. Also put a close box in every window, which works in the usual way.

Unit 14. Implement selection in various ways (by one click; by a click and then a shift-click; by a click and then a drag). Implement highlighting for selection (use yellow). When you select something, move it to a clipboard area. Add to your dialog box a Show Clipboard choice. Add to your dialog box a Select All choice.

This is most important for a word processing program, because large areas of text can be selected, but it can also be used in a spreadsheet or a data base program. The Select All choice, indeed, is optional for spreadsheets and data bases, and might be replaced by something which merely selects one entire cell in a spreadsheet, or one entire record in a data base.

Unit 15. Implement deletion of a selection. This will imply deletion from your internal data structure and then redrawing. Add to your dialog box a Cut choice, which deletes a selection. For a word processing program, the redrawing applies to the entire document; for a spreadsheet, it applies merely to one cell; for a data base, it applies merely to one field within one record. Ways of deleting several cells from a spreadsheet, or one or more entire records from a data base, might also be implemented.

Unit 16. Implement paste. This will imply insertion into your internal data structure and then redrawing. Add to your dialog box a Paste choice, which inserts a selection. Redrawing is as with Cut.

Unit 17. In the end, a GUI application is not realistic without Undo; and yet Undo is one of the hardest functions to implement, and takes the most time. There is single-level Undo and unlimited Undo, each with its own techniques. A single-level Undo function is generally accompanied by a Redo function, which "undoes the Undo." Remember that you can't undo a file operation like opening, saving, or printing. As a first cut at this, add to your dialog box an Undo choice, but, when the user undoes anything, just bring up an alert box that says "You cannot undo this" and lets the user click OK.

Unit 18. Rewrite as many of the earlier functions as you can, to implement the single level Undo. These include:

• For deletion of a selection: Delete it but save it somewhere, together with an indication of where you deleted it from. To undo, just put it back in.

• For pasting: Paste but remember what it looked like before pasting. To undo, just set it back again.

• For typing: This is a special case of insertion. If a selection was highlighted just before you started typing, then, when the typing is undone, not only is it taken out but the previously highlighted selection is also put back in.

Unit 19. Find out how the print-window function works in your GUI. You might have to break your image up into pages by itself, to get it printed. Find out how the system print dialog works. Add a Print button to your dialog box, to print the currently active window (or the window with the focus).

For a word-processing program, if you have implemented vertical scrolling, then the entire document may be broken up into pages, and each page printed separately. You might, at this point, wish to implement a page-break facility.

For a data base program, each page is either one or more entire records in the data base. Do not break up a record across pages unless one record is so large that it will not fit on a page.

For a spreadsheet program, there is a problem if there are so many columns that they will not all fit on a page. In that case, do only the first few columns first, then the next few, and so on. Repeat the process if there are too many rows to fit on a page.

Unit 20. Set up a menu bar. It should have File, Edit, and Window menus. The Window menu should allow you to make any window the current window. The Edit menu should have Undo, Cut, Copy, Paste, and Select All. The File menu should have New, Open, Close, Save, Save As, Print, and Quit. Now use this instead of the dialog box that you have been using so far.

Unit 21. Find out how your GUI implements key equivalents of menus (if at all). Put in some key equivalents that resemble others that are common on your GUI.

Unit 22. Implement a sort function for your data. Put a Sort item in your Edit menu. Remember to redraw the window after you sort. If nothing is selected, sort the whole file; if something is selected, sort only that part.

Unit 23. Implement graying and ungraying of the Save, Cut, and Copy menu items. To gray an item means to make it appear on its menu in gray, which disables it, so that it cannot be selected. To ungray an item means to make it appear in black again, so that it is reenabled. This is done as follows:

For Save, the item should be grayed as soon as a Save takes place. It is ungrayed when any further change is made to the document.

For Cut and for Copy, the item should be grayed if there is no highlighted selection; it should be ungrayed when a selection is highlighted.

Unit 24. Demonstrate your project.