ECE 118 LAB EIGHT

This week you are going to make an interactive calculator. It doesn't have to be very fancy, just functional. Like the kind you could buy for $10.

1. *A Button*

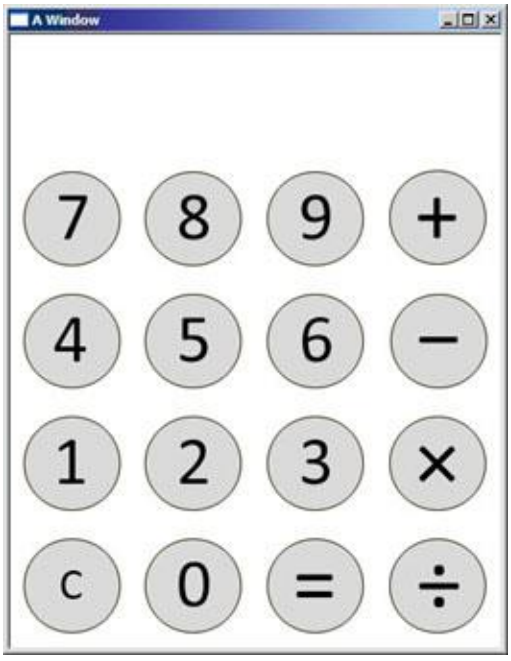
Write a function that will draw a single button, of the sort that might be useful for a calculator. Make it stylishly round. You may have to experiment with the pen position to get the digits properly centred.

The function should be given parameters to tell it the size of the button and what symbol it should contain within it. Shade the button and give it an outline just so that it doesn't look dreadful.

You may like to remember that there is a library function called `set_font_size(`*n*`)`. It selects the font to be used by `write_string` so that it will fit neatly in a box n pixels high. What a coincidence.

2. *Some Buttons*

Now write a function that draws a whole grid of buttons, as they would appear on a calculator. Remember to leave space for a numeric display to be added later.



Plan the positions of your buttons carefully so that they fit neatly within the window. You should define named constants for the window's width and height, and calculate button sizes and positions from those values. Then you will be able to change the size of your calculator at any time, without having to recalculate everything. Eventually you should even make the font size depend on the window size so that it always looks right, but that will require a little experimentation.

Take care to ensure that you have a simple regular calculation for the positions of buttons, otherwise the next step will be unnecessarily complicated.

A note about the special characters × *and* ÷.

They don't appear on the keyboard, and the way of typing them that used to work suddenly doesn't. The only method I have found is to copy and paste them from another document, such as this one. To display them, you have to use the special function `write_char`, they must be written as a single character in single quotes preceded by an `L`, and if they are stored in a variable, array, or whatever, the declaration must be `int`. This shows all of that:

```
const int buttons[4][4] =
{ { L'7', L'8', L'9', L'+' },
  { L'4', L'5', L'6', L'-' },
  { L'1', L'2', L'3', L'×' },
  { L'c', L'0', L'=', L'÷' } };

void main()
{ make_window(700, 700);
  /* whatever */
  write_char(buttons[3][3]);
  /* whatever */ }
```

3. *Clicking*

The graphics library allows your program to detect mouse clicks. This little snippet of code:

```
wait_for_mouse_click();
const int x = get_click_x(), y = get_click_y();
cout << "Mouse clicked at position (" << x << ", " << y << ")\n";
```

causes a program to wait until the mouse is clicked somewhere within its graphics window, then report the co-ordinates of the pixel that was clicked on. Try it out. Put that in a loop after you've drawn the grid of buttons, and make sure it does what you would expect.

Now the real task is to convert the pixel co-ordinates to something that represents which button (if any) the mouse was clicked within. If you chose the same layout of buttons as I did, and you have a simple calculation for the positions of buttons, this will be easy.

For now, just work out which row and column of buttons the mouse click was in. In the diagram, the "7" button is in row 1 column 1, and the "×" button is in row 3 column 4. So your program should be modified so that a mouse click on the "7" button makes it print "`clicked row 1 column 1`" and a click on the "×" button makes it print "`clicked row 3 column 4`", and so on. Then don't forget to check that the click was actually inside the circle of the button.

4. *What did you click on?*

Now convert that bit of code into a very useful function. Whenever it is called, the function should wait until the mouse is clicked, and work out which row and column of buttons the click corresponds to, exactly as before. After that, it should return as its result a value indicating the label of that button. Perhaps 0 to 9 for the digits, and other numbers for the other symbols. A `char`-typed variable may be helpful.

There is no clever trick to work out for this. Once you know the row and column numbers, the best plan is probably just to have a bunch of ifs, one for each button, returning the right label. So if the click was on row 2 column 2, this function should return `'5'`.

Put it all in a little program and test it well.

5. *Almost a calculator*

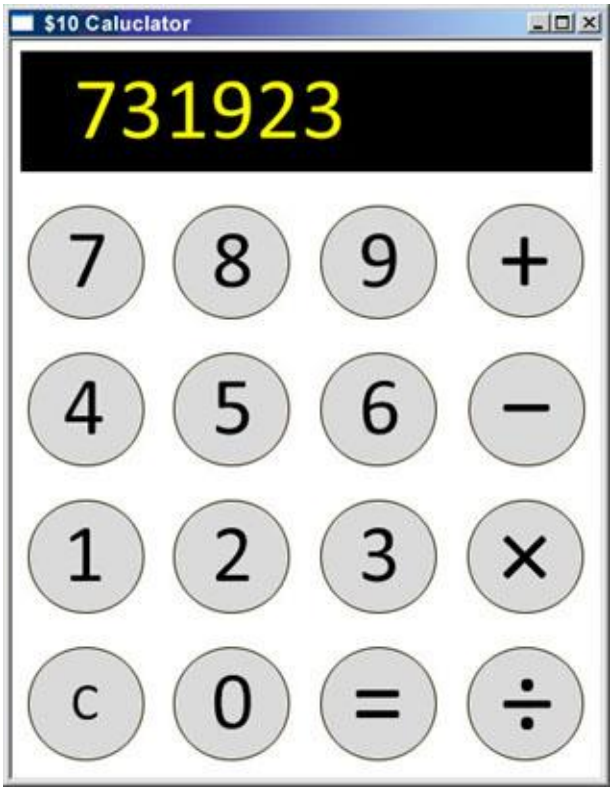
Just for now, ignore the + - × ÷ = buttons, and concentrate only on the numbers and the Clear button. As each digit button is clicked, your program should keep track of the entire number that has been entered (so if `'7'` then `'4'` then `'8'` are clicked, it should have in its mind the int value 748).

There is an easy way to do this, and it is one of those situations where a variable is useful. If your program allows itself to have an int variable, initially set to zero, that will accumulate a number as it is entered, this plan will work:

repeat this loop:
- Wait for a button to be pressed.
- If it was the `'9'` button, multiply the variable by 10 and add 9.
- If it was the `'8'` button, multiply the variable by 10 and add 8.
- If it was the `'7'` button, multiply the variable by 10 and add 7, *etc etc etc.*
- If you are using a char value, and returning (for example) `'5'` from the button detector, what you are actually receiving is an ASCII code, in which the char `'5'` is represented by the number 53. Subtracting `'0'` from it will turn it into the desired number, 5.
- Clear should reset absolutely everything in your program, that will make debugging much easier.
- Update the display to show the new value of the variable.

"What display?" you may ask. That of course is what the empty region at the top of the window is for. The library function `write_string` will happily take an `int` as its parameter.

Make it happen.



6. *Finally*

Make the other five buttons do their thing.

Get a clear idea of exactly what should happen when each button is pressed. It is not complicated, but unless you think it through first, your program might be. You can keep it restricted to just `int`s, but try to make it into a sturdy product, one that doesn't explode if you try to divide something by zero for instance.