EEN118 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*

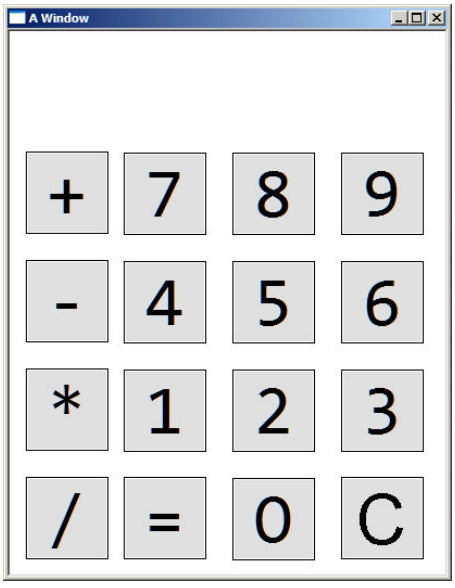
Write a function that will draw a single button, of the sort that might be useful for a calculator

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.



It is absolutely vital that the `+`, `-`, `*`, and `/` buttons are at the left, as shown.

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.

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 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 2, and the "`*`" button is in row 3 column 1. So your program should be modified so that a mouse click on the "`7`" button makes it print "`clicked row 1 column 2`" and a click on the "`*`" button makes it print "`clicked row 3 column 1`", and so on.

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 the some value indicating the label on that button.

There is no clever trick to work out for this. Once you know the row and column numbers, one possible plan would be to have a bunch of `if`s, one for each button, returning the right label. So if the click was on row 2 column 3, this function would return 5 or the string "5" or the character code '5', or something like that.

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. 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 can be 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:

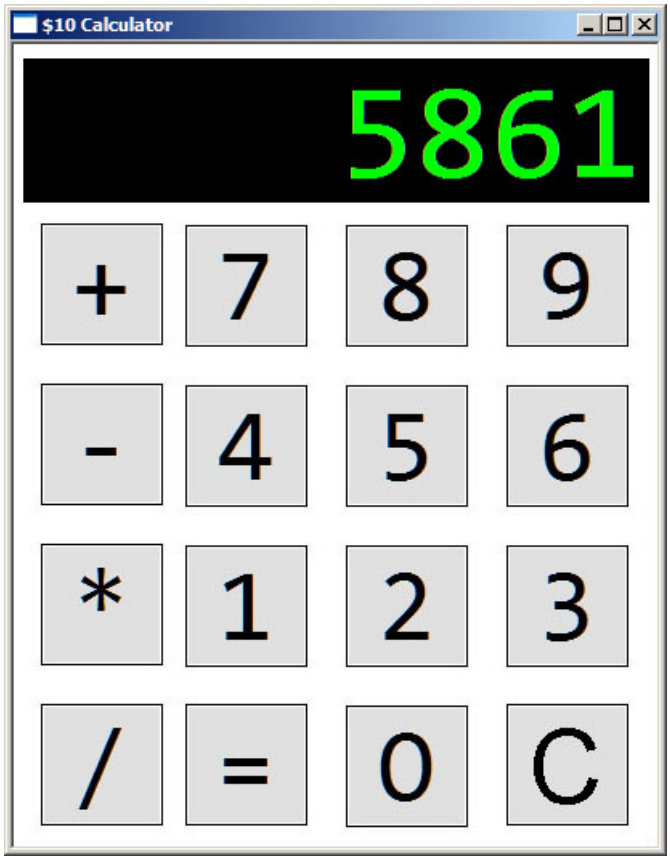
repeat this loop:

Wait for a button to be pressed.
If it was the "9" button, the multiply the variable by 10 and add 9.
If it was the "8" button, the multiply the variable by 10 and add 8.
If it was the "7" button, the multiply the variable by 10 and add 7, *etc etc etc.*
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.

But of course, you should do things one at a time. First just print the value in the normal way in the text window. When you know you are calculating the right values consistently, then start `write_string`ing it on the display.

Make it happen.



6. *Calculate!*

Make the other 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.

What must happen when you key in the sequence 1 2 3 + 2 1 × 7 5 = for example? It starts out just reading a number as for part 5, but when the + is entered something extra must happen. It can't do the addition just yet, it must read another number first. While the second number is being read, the calculator must remember what the first number was, and the fact that an addition is still to be done.

Whenever a non-digit is encountered, if there is a pending operation, it is used to combine the two numbers to produce a new "old" number. With a few details left for you to work out, it just continues like that.

Your program should print all the variables after every key click. Then if anything goes wrong you will already be half way to knowing what it is.

6. *Useful.*

Make sure that you have got a calculator program, with the buttons in the places demanded by part 2, ready to turn in.

Then think what you would like a calculator to be like. If you kept the design simple, you can make your own version that behaves exactly as you would like it to, with as many fancy buttons as you like (the lab guys can tell you how to make interesting symbols appear on the keys). Fed up working out the logarithm-base-two of things? Give yourself a $\log_2$ button! Troubled by pesky quadratics? Make it solve them for you. Adding your own operations is as easy as falling out of a tree. If you do something really good, you can submit it too, and maybe get a little extra credit.