ECE118 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 lab guys will tell you of a trick for getting the labels properly centred.

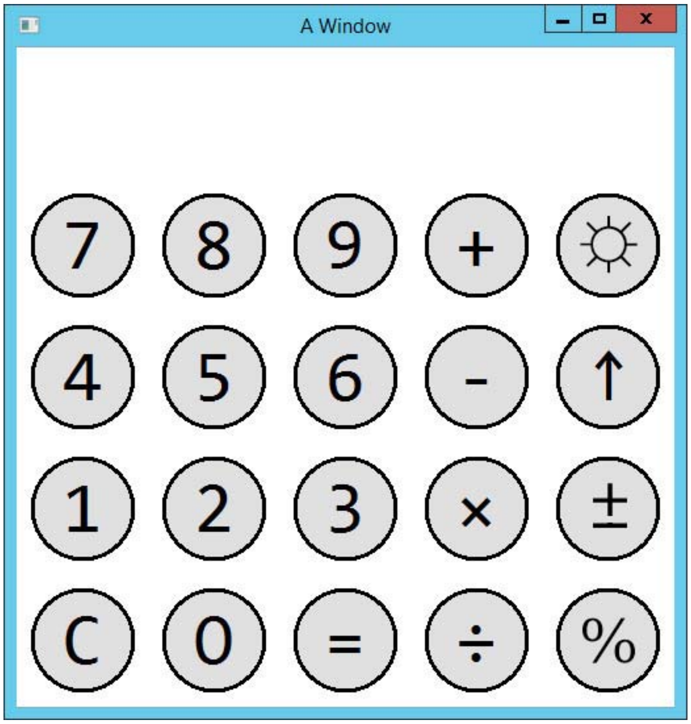
➡ We want round buttons, so that they won't hurt anyone in an accident.

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 size of the font to be used by `write_string` or `write_char` 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.



You are probably wondering just what the fifth column of buttons are supposed to represent. ☼ is the power on/off button. ↑ stands for "to the power of", although that isn't the way it is written in C++. ± changes the sign of the result between positive and negative. % is the normal remainder/modulo operator from C++.

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.

About special characters.

× and ÷ don't appear on the keyboard, but don't let that worry you. A program like word will allow you to type all sorts of strange symbols (insert → symbol), and once the symbol you want is in a word document, you can copy and paste it into visual studio. If it's available, the windows program "character map" will let you insert the weirdos straight into your C++ without having to go through word at all.

But beware! Do not use `write_string` for anything you can't type normally on the keyboard. Use `write_char` instead. It can only print a single character at a time, but it can print anything. Use it like this:

```
write_char(L'÷');
```

That's right, there is a capital L in there, and there are only single-quotes around the character. That's how you type a *unicode* character constant in C++. Something like this might be even better:

```
const int divide_sign = L'÷';
// ... and later ...
write_char(divide_sign);
```

There is a potential problem. It is possible that for editing your C++ program Visual Studio will use a font that doesn't contain all the weird characters. If that happens, it will give you an odd error message about saving your code, and you will have to use this work-around instead:

```
write_char(L'\u2191');  // produces ↑
write_char(L'\u263C');  // produces ☼
```

If all else fails, you can just pick another character that you like the look of.

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. 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.

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 `if`s, 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. *Entering Actual Numbers*

Just for now, ignore all the buttons except for the numeric ones and Clear. As each numeric 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 that will accumulate a number as it is entered, this plan will work:

```
start with the variable set to zero.
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.
    Print the new value of the variable.  }
```

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.

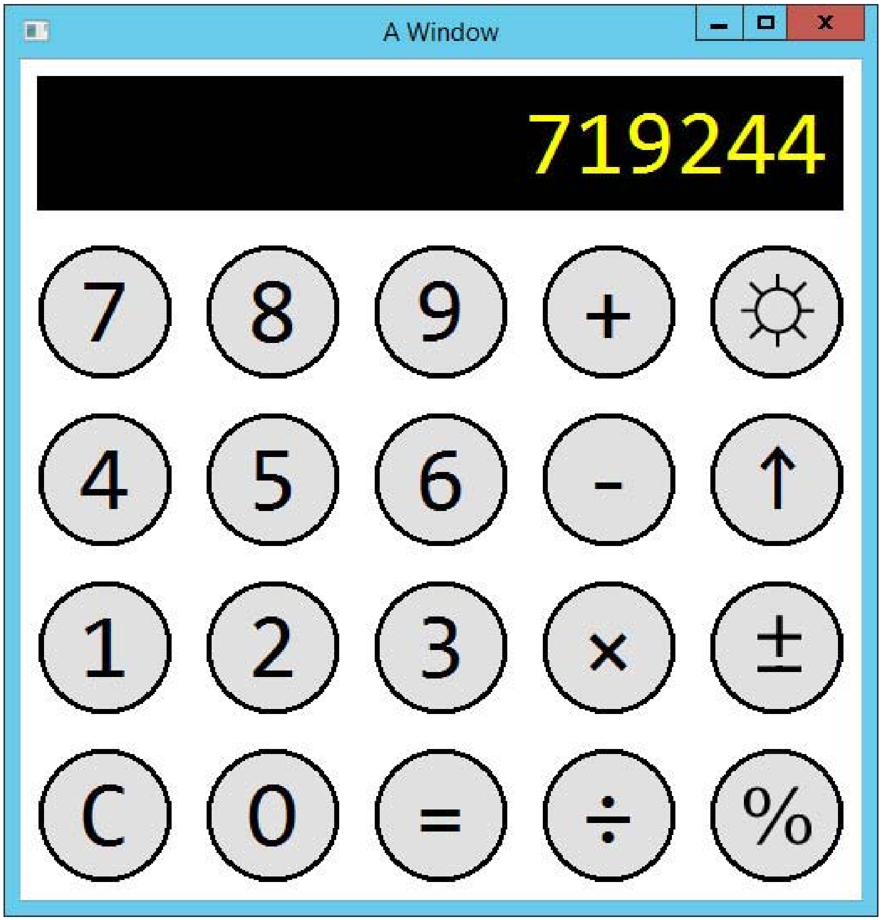
What must happen when you key in the sequence 1 2 3 + 2 1 × 7 5 = for example? It starts out just reading a number as for part 5, but when the + is entered something extra must happen. It can’t do the addition just yet, it must read another number first. While the second number is being read, the calculator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

The trick is to give yourself a couple of variables. One remembers the number currently being read, *à la* part five. Another remembers the pending operation, +. Another remembers the old number that has already been entered.

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 is wrong.

7. *Finally, an Actual Calculator*



There isn't much to say about the display, just make it always show the current value if the power is on (remember the '☼' power on/off button?). The library function `write_string` will happily take an int as its parameter.

Test your program carefully, this could be something useful to have. Yes, windows and everything else comes with a calculator built in, but now you can have a calculator that does whatever you need it to do, not just what microsoft thinks you need to do. Adding your own operations is as easy as falling out of a tree.