Handout 11

The `for` loop

11.1 The `for` construction

Consider this code:

```
k=2;
while(k<=7)
{
  printf("Counter is %d\n",k);
  k++;
}
```

Recall what this does:

1. Set k equal to 2
2. Checks if $k \leq 2$. If not, exits. If so, proceeds to stuff in curly braces.
3. Does the commands inside the curly braces, ending with increasing the value of k.
4. Returns to step 2.

We can accomplish these same steps with a `for` loop:

```
for(k=2; k<=7; k++)
{
  printf("Counter is %d\n",k);
}
```

Important: the `k++` is done at the *end* of the stuff in the curly braces!
In general, if the number of loops is easily predicted, `for` loops are a good option.

11.2 `for` loops and arrays of integers

Write a program that constructs an `int` array with 100 slots, filled as follows:

1
2
3
⋮
100

1. Add another `for` loop to your code in order to compute $1 + 2 + 3 + \cdots + 100$.
2. Then modify this loop in order to compute $1 - 2 + 3 - 4 + \cdots - 100$.

11.3 `for` loops and arrays of doubles

Construct a `double` array that contains the following

1
$\frac{2}{3}$
$\frac{4}{9}$
$\frac{8}{27}$
⋮
$\frac{2^{100}}{3^{100}}$

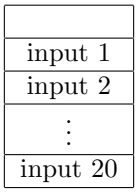
1. Use the array (a `for` loop) to compute

$$1 + \frac{2}{3} + \frac{4}{9} + \frac{8}{27} + \cdots + \frac{2^{100}}{3^{100}}$$

2. Then modify your code so that it requests two integers n_{start} and n_{end} from the user. The output of the code is
   - the sum
   $$\frac{2^{n_{\text{start}}}}{3^{n_{\text{start}}}} + \cdots + \frac{2^{n_{\text{end}}}}{3^{n_{\text{end}}}}$$
   - If the values of n_{start} and n_{end} make it so that you cannot use your array to do the computation, in which case tell the user the input won't work.

11.4 Homework: `list-min.c`

Write a piece of code that first takes in a list of 20 integers (use `int`!) and puts them in an array as follows:



Then have your code search through the array and find the smallest value. Have your code put this value in the first slot in the array.

Finally, have your code print out the entries of the entire array.

For example, if I give you the input

```
10 9 8 7 6 5 4 3 2 1 11 10 9 8 7 6 5 4 3 2
```

then your code will print out the list

```
1 10 9 8 7 6 5 4 3 2 1 11 10 9 8 7 6 5 4 3 2
```

11.5 Challenge problem

Write a program that takes in a list of integers greater than or equal to zero. The list ends with the integer -1. You don't know how many integers will appear on the list, but you want to stop accepting inputs once you see -1. You can't expect to store all of the input because you cannot predict how much memory to set aside. But what you can do is store the most recent 10 numbers. (This problem is a "toy model" for things like security cameras that only store the most recent data.)

Build a piece of code that takes in the list of integers as described, and then prints out the most recent 10 integers. (Before you start writing code, think carefully about what your process is going to be!)