The Preprocessor

With preprocessor macros, you can make the C language look rather different. In fact, many of the horrible tricks that have been used in the International Obfuscated C Code Contest depend on using and abusing the preprocessor. For this exercise, we're going to over-use it a little bit. Please don't use techniques like this on you projects. It will make it harder to grade your work, especially the parts pertaining to style.

Consider the following program. It reads a list of doubles from standard input, using a resizable array to store as many values as needed. Then, it sorts the array and prints the values out in sorted order:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  // Make a resizable list.
  int list_cap = 5;
  int list_len = 0;
  double *list = (double *) malloc( list_cap * sizeof( double ) );

  double val;
  while ( scanf( "%lf", &val ) == 1 ) {
    // Grow the list when needed.
    if ( list_len >= list_cap ) {
      list_cap *= 2;
      list = (double *) realloc( list, list_cap * sizeof( double ) );
    }

    // Add this item to the list
    list[ list_len ] =val;
    list_len++;
  }

  // Bubble-sort the list.
  for ( int i = 0; i < list_len; i++ )
    for ( int j = 0; j < list_len - i - 1; j++ )
      if ( list[ j ] > list[ j + 1 ] ) {
        double tmp = list[ j ];
        list[ j ] = list[ j + 1 ];
        list[ j + 1 ] = tmp;
      }

  // Print out the resulting, sorted list, one value per line.
  for ( int i = 0; i < list_len; i++ )
    printf( "%.2f\n", list[ i ] );

  return EXIT_SUCCESS;
}
```

On the course homepage, you'll find a source file, sortList.c, along with a sample input and expected output. You can download these files from the web page or with the following curl commands:

curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise17/sortList.c
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise17/input.txt
curl -O https://www.csc2.ncsu.edu/courses/csc230/exercise/exercise17/expected.txt

If you look in the file, sortList.c, you'll see a program that looks a little like this, but I've replaced many parts of the program with macros. Your job is to add macro definitions at the top of the file so that the macros used in this program expand to the code given above (probably with some differences in things like spacing, newlines and semi-colons). For example, instead of writing out code for the declarations used in a resizable array, the sortList.c program uses the macro:

```
DECL_LIST( double, list );
```

This should expand to the following code (with some differences in spacing and newlines).

```
  int list_cap = 5;
  int list_len = 0;
  double *list = (double *) malloc( list_cap * sizeof( double ) );
```

Notice that this uses the first parameter to specify the type of the resizable array and the second parameter to determine the name of the array and its associated variables, list_cap and list_len. This would let us use the same macro to help implement any kind of resizable array we need. This technique was used before we had C++ and its template syntax (like generics in Java).

To build names like list_cap and list_len in the macro expansion, you'll need to use the ## operator in your macro definition. Remember, this lets you connect the value of a parameter with other syntax, without any spaces in between. So, for example, if you defined a macro like:

#define DINO(x) x##osaurus

And used it in the following:

```
DINO( bront )
DINO( all )
DINO( ankyl )
```

It would expand to:

```
brontosaurus
allosaurus
ankylosaurus
```

To complete this exercise, you'll need to define the following macros:

- DECL_LIST(type, name)
  Expands to the declaration syntax for the three variables used in a resizable array implementation, name_len, name_cap and name (the pointer itself).
- CHECK_CAP(type, name)
  Expands to the code to make sure the resizable array has enough capacity for another element and to enlarge it if it doesn't.
- SIZE(name)
  Expands to an expression for the number of elements currently stored in the array.
- FOR(var, limit)
  Expands to the first line of a for loop that uses an int variable named var to iterate from 0 up to (but not including) limit.
- SWAP(type, a, b)
  Expands to a block of code that swaps the values of two variables a and b of the given type. We need the type parameter in order to declare a temporary variable that will help swap the values.

As you refine your macro definitions, remember that you can use the -E command-line option to see what your code looks like after preprocessing. This can help you see and fix errors that result from macro expansion (pretty much the only errors you're likely to get in this exercise).

Once you have the needed macro definitions, you should be able to compile and run your program and it will work just like the sample code shown at the start of this exercise. Submit your source code to the exercise_17 assignment on Moodle. But, remember, don't start defining macros like FOR( ) in your homeworks. This will make it hard to evaluate your style.