Lab 9: variadic functions in C, C++

- variadic functions accept a variable number of arguments (e.g. printf)
- in C this is done with macros from stdarg.h, and example is provided in varArgs.c in the lab9 repository
- in C++ this is usually done with templated functions, an example is provided in template.cpp in the lab9 repository
- for the lab9 exercise you'll be implementing a "PairSums" function, once in C (lab9.c) and once in C++ (lab9.cpp)
- as usual, details are in the repository README, and a makefile is provided for compilation of the various files

The PairSums function

- To accommodate the C macros, the PairSums function is specified slightly differently in the C/C++ versions
- In both cases, it adds and displays pairs of numeric arguments passed to it, e.g. in the C++ version

  PairSums(5.4, 6, 10, 1.3) prints 11.4 and 11.3

  (the sum of the first pair, and the sum of the second pair)

  If an odd number of values is passed then it treats the “missing” last value as 0
- In the C version you need an extra (first) parameter, specifying how many other values are being passed, e.g. for the same data as above the C call would look like

  PairSums(4, 5.4, 6, 10, 1.3)

C version (stdarg.h macros)

- The set of optional arguments is represented using ... in the function declaration, e.g.

  void PairSums(int numArgs, ...)

- To access and initialize the list of arguments within the function, you create a variable of type va_list and initialize it with function va_start, e.g.

  va_list L;

  va_start(L, numArgs);

- To get the first/next argument from the list, use va_arg, specifying the expected type

  double d = va_arg(L, double);

- va_arg accepts only a limited set of data types: longs and doubles are safe
- At the end of the function, delete the list of args using va_end, e.g.

  va_end(L);

C++ version (templates)

- For variadic functions using templates, we provide a version of the function that accepts just one parameter (a base case), plus a version of the function that accepts more than one parameter (a general case)
- The general case processes the first N arguments it is passed, e.g. the first 2 in the case of PairSums, then recursively calls itself to process the rest

C++ Sum example (base case)

- for a function to take the sum of an arbitrary number of arguments, the (templated) base case might look like

```
Template <typename T>
T Sum(T x) {
    return x;
}
```

- For PairSums you'll need two parameters

C++ Sum example (general case)

- For our Sum example, the general case needs to specify the first argument, and a second parameter with special syntax to specify the rest
- It then needs to express the result as some process using the first argument and the result of a recursive call, e.g.

```
template<typename T, typename... Args>
T Sum(T front, Args... args) {
    return front + Sum(args...);
}
```

- Note the way ... is used in all three locations and the template syntax used to declare the variable collection of arguments
- For PairSums you'll need two parameters before the Args list