1. Normally, function bodies do not appear in header files, but they do appear in the header file for our template of dynamic arrays. Why?

Templates are not class and member functions (which can be compiled separately), but rather are instructions for the compiler to instantiate a new class given (or deducing) a type. The compiler, when coming across this template, needs to have access to the implementation of the class and its methods to properly instantiate them with the template argument. If these template class implementations weren’t in the header file, such would be inaccessible. The keyword export did make it possible to allow separate implementation files but due to scare adoption by vendors, it has since been deprecated.

1. What are the differences between templates and macros?

Macros serve as text substitution tools during preprocessing whereas templates are executed at compile time and become type-aware and scoped. Templates take longer to compile, but are safer. I did expand on this point in the discussion section:

We can define the following: #define tempmax(a,b) ((a)>(b)) ? (a) : (b)) and then use int top = max(C++,F) which will simply substitute "C++" (Celsius) into the macro (where a is) but if we were to use a template:

template <class T> tempmax (T a, T b) {return (a>b) ? a : b)}

Then the compiler will extract variable information from Celcius and not just replace it in the position where a is as in the preprocessed macro. This would take longer as the argument has to be parsed during compilation, becoming type-aware and scoped. If these two concerns are not an issue and speed is the name of the game, a macro could work just fine.

1. Discuss the differences between formal parameters to a function and the parameters to a template.

C++ templates allow one to implement a generic Class<T> template that has a type parameter T. T can be replaced with actual types that will generate the class Class<myType>. One could specify a default template parameter like in function parameters; if you specify a default template parameter for any formal parameter, the rules follow from functions and default parameters. The parameter type in a template (usually “T”) cannot be redefined in the scope of the template declaration, unlike in function declarations (else would get an error for redefining a type-parameter). You also cannot modify a non-type-parameter from the template arguments. It is also important to note that templates are not member functions as discussed previously and are compiled as tools to instantiate classes for a specific set of types.