1. An alternate way of implementing general linked lists is with templates. What would the parameter(s) to the template be?

Parameter types could be T data, const/non-const T &data, T\* data, and an iterator variable one could name iterator to, and iterator from (or have it just be iterator itr).

1. What are the tradeoffs between implementing lists using inheritance per the example, and implementing them using templates?

Template definitions must be visible on every translation unit, so definitions must not be placed in a separate cpp. They must be present the header file. Unlike with templates, in MI polymorphic function inheritance brings with it the challenge of not having universally-quantified types; that is, pure virtual functions and derived classes from an ADT have to be redefined to manipulate certain data types. You are not afforded the luxury of being ignorant about the implementation with MI. Another crafty tool a programmer can use with C++ templates is for "template metaprogramming," which means writing pieces of code that run while the main program gets compiled rather than when it runs. While I have only briefly read about this, it seems to indicate that the speed of the linkedlist program can be improved by focusing resources in compilation phases (by generating temporary source code later merged with the rest of the source code *then* compiled)-- by getting the compiler to focus in on certain portions of code. In a way templating hides some of our intentions from the compiler when compared with the inheritance approach, but it becomes much more versatile for it.

1. Is the generation number a foolproof way of preventing the use of stale iterators? Can you devise a better way of doing this? A foolproof way?

You could go about this by creating your own comprehensive iterator class using templates to avoid coming up against stale iterators; another is resorting to the use of std::list<T> or the std::unique\_ptr that manages another object through a pointer and disposes of that object when it goes out of scope.